Semantics and Planning Based Workflow Composition and Execution for Video Processing

Gayathri Nadarajan

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh

2010

Abstract

Traditional workflow systems have several drawbacks, *e.g.* in their inabilities to rapidly react to changes, to construct workflow automatically (or with user involvement) and to improve performance autonomously (or with user involvement) in an incremental manner according to specified goals. Overcoming these limitations would be highly beneficial for complex domains where such adversities are exhibited. Video processing is one such domain that increasingly requires attention as larger amounts of images and videos are becoming available to persons who are not technically adept in modelling the processes that are involved in constructing complex video processing workflows.

Conventional video and image processing systems, on the other hand, are developed by programmers possessing image processing expertise. These systems are tailored to produce highly specialised hand-crafted solutions for very specific tasks, making them rigid and non-modular. The knowledge-based vision community have attempted to produce more modular solutions by incorporating ontologies. However, they have not been maximally utilised to encompass aspects such as application context descriptions (*e.g.* lighting and clearness effects) and qualitative measures.

This thesis aims to tackle some of the research gaps yet to be addressed by the workflow and knowledge-based image processing communities by proposing a novel workflow composition and execution approach within an integrated framework. This framework distinguishes three levels of abstraction via the design, workflow and processing layers. The core technologies that drive the workflow composition mechanism are ontologies and planning. Video processing problems provide a fitting domain for investigating the effectiveness of this integrated method as tackling such problems have not been fully explored by the workflow, planning and ontological communities despite their combined beneficial traits to confront this known hard problem. In addition, the pervasiveness of video data has proliferated the need for more automated assistance for image processing-naive users, but no adequate support has been provided as of yet.

A video and image processing ontology that comprises three sub-ontologies was constructed to capture the goals, video descriptions and capabilities (video and image processing tools). The sub-ontologies are used for representation and inference. In particular, they are used in conjunction with an enhanced Hierarchical Task Network (HTN) domain independent planner to help with performance-based selection of solution steps based on preconditions, effects and postconditions. The planner, in turn, makes use of process models contained in a process library when deliberating on the steps and then consults the capability ontology to retrieve a suitable tool at each step.

Two key features of the planner are the ability to support workflow execution (interleaves planning with execution) and can perform in automatic or semi-automatic (interactive) mode. The first feature is highly desirable for video processing problems because execution of image processing steps yield visual results that are intuitive and verifiable by the human user, as automatic validation is non trivial. In the semi-automatic mode, the planner is interactive and prompts the user to make a tool selection when there is more than one tool available to perform a task. The user makes the tool selection based on the recommended descriptions provided by the workflow system. Once planning is complete, the result of applying the tool of their choice is presented to the user textually and visually for verification. This plays a pivotal role in providing the user with control and the ability to make informed decisions. Hence, the planner extends the capabilities of typical planners by guiding the user to construct more optimal solutions. Video processing problems can also be solved in more modular, reusable and adaptable ways as compared to conventional image processing systems.

The integrated approach was evaluated on a test set consisting of videos originating from open sea environment of varying quality. Experiments to evaluate the efficiency, adaptability to user's changing needs and user learnability of this approach were conducted on users who did not possess image processing expertise. The findings indicate that using this integrated workflow composition and execution method: 1) provides a speed up of over 90% in execution time for video classification tasks using full automatic processing compared to manual methods without loss of accuracy; 2) is more flexible and adaptable in response to changes in user requests (be it in the task, constraints to the task or descriptions of the video) than modifying existing image processing programs when the domain descriptions are altered; 3) assists the user in selecting optimal solutions by providing recommended descriptions.

Acknowledgements

Conducting research in an interdisciplinary field has been demanding, yet rewarding. I would like to thank my supervisors, Jessica Chen-Burger and Bob Fisher for their guidance while undertaking this challenging journey. Jessica for introducing this fascinating problem domain for me to research into and the many ideas that she has given me to explore. Bob's wisdom and constant emphasis on the importance of storytelling have helped turned this into a coherent, readable thesis. To my examiners, Omer F. Rana and Dave Robertson, thank you for the positive and encouraging viva and the reflective insights on the research undertaken. I'm also indebted to Björn Franke, Maciej Zurawski and Gerhard Wickler for proof-reading chapters or drafts of the thesis.

Much gratitude goes to Concetto Spampinato from the University of Catania, Italy for the video processing collaborations, in particular the many visits to Edinburgh and for hosting me in Sicily during the autumn of 2008, Arnaud Renouf from GREYC, France for collaboration on the ontologies, Andrew Grimmer from the School of Biological Sciences for the useful feedback on the prototype that I built, Annabel Harrison for assisting with user learnability issues and all experiment participants. I especially thank my colleague Shahriar Bijani for the stimulating ideas and discussions.

Thanks are also due to my friends who have put up with me whether they realise it or not; David Spark, Chris Fensch, Robert Block, Matthew Whitaker, Christophe Dubach, Jorge Adriano, June Tee, Mariola Mier, Marisa González, Li Wen Kuok, Wai Leng Say, Yun Mi Hwang and Zhenkun Wang. Thanks to my enthusiastic students from the robots-playing-football SDP module that thesis writing was less daunting, and to the Taekwondo club crew for the bouts in the *dojang* that kept me sane.

My parents Soshila and Nadarajan have always believed in me, my nephews Danial and Harith, and niece Dania have kept me entertained, captivated and inspired by their boundless energy, imagination and innocence. No words can express my gratitude to my sister Shuba for her unconditional patience, support and love throughout.

I'm grateful to several institutions and individuals who have provided financial assistance, work and travel opportunities throughout my studies, which was otherwise self-funded; Informatics Teaching Organisation, Informatics Graduate School, CISA (Alan Smaill), the OpenKnowledge project, the University of Catania (Concetto Spampinato), Google Anita Borg Scholarship Europe, Jane Lockhart, IDIAP research institute, the SynProc project, ICAPS and IJCAI conferences. Final thanks go to NCHC, Taiwan for providing access to the Ecogrid videos in order for this research to be conducted and further explored in the forthcoming Fish4Knowledge project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. The following are the principal publications derived from this thesis:

- G. Nadarajan, Y.-H. Chen-Burger, R. B. Fisher. "A Knowledge-Based Planner for Processing Unconstrained Underwater Videos". In *IJCAI Workshop on Learning Structural Knowledge From Observations (STRUCK 09)*, July 2009.
- G. Nadarajan, A. Manataki, Y.-H. Chen-Burger. "Semantics-Based Process Support for Grid Applications". Book Chapter in Nik Bessis (ed.): *Grid Technology for Maximizing Collaborative Decision Management and Support: Advancing Effective Virtual Organizations*, IGI Global, May 2009.
- C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, R. B. Fisher. "Detecting, Tracking and Counting Fish in Low Quality Unconstrained Underwater Videos". In *International Conference on Computer Vision Theory and Applications (VISAPP 08)*, Jan 2008.
- G. Nadarajan. "Planning for Automatic Video Processing using Ontology-Based Workflow". Doctoral Consortium at *International Conference on Automated Planning & Scheduling (ICAPS 07)*, Sept 2007.
- G. Nadarajan and A. Renouf. "A Modular Approach for Automating Video Analysis". In *International Conference on Computer Analysis of Images and Patterns (CAIP 07)*, Aug 2007.
- G. Nadarajan, Y.-H. Chen-Burger, J. Malone. "Semantic Grid Services for Video Analysis". In *International Workshop on Service Composition*, Dec 2006.
- G. Nadarajan, Y.-H. Chen-Burger, J. Malone. "Semantic-Based Workflow Composition for Video Processing in the Grid". In *IEEE/WIC/ACM International Conference on Web Intelligence (WI 06)*, Dec 2006.

(*Gayathri Nadarajan*)

To my family

To those who strive to outdo themselves with passion, strength and courage

Table of Contents

Chapter 1

Introduction

"A huge gap exists between what we know is possible with today's machines and what we have so far been able to finish."
Donald Knuth

Intelligent systems have a long standing reputation for attempting to solve AI problems such as knowledge representation and reasoning, learning, planning, communication and perception. A few classical examples include expert systems, robots and game-playing agents. In the last twenty years or so, AI systems enjoyed their greatest success with the increase in computing power and also when integrated with other fields, such as logistics, data mining and medical diagnosis. There has been an abundance of intelligent systems that help humans each day in performing tedious tasks more efficiently than they would with traditional manual means. The existence of such systems has also led to the need for developing more human friendly interfaces for non-technical users to manipulate without being equipped with the technical know-hows of the complex problem solving environment. As is evident now, even non-technical users are dependent on technology and computing applications in their day-to-day activities, in particular domain experts, *e.g.* medical practitioners, earth scientists, physicists, *etc*. While various computational tools exist to assist such users with their everyday tasks, such as workflows, there is still a gap when mapping their high level requests to the corresponding low level computations. This problem is exacerbated when the users do not possess the technical expertise to choose the combination and sequence of tools that will provide the best solution. One complex domain where such adversities are exhibited is video processing.

Video and image processing problems have become part and parcel of many prac-

tises today. The field of video analysis is becoming more and more important with the fast advancement in computer vision technologies and the increasing size of real-time data that need to be processed. The pervasiveness of video data today, *e.g.* satellite images, surveillance videos and environmental monitoring videos, has triggered the need for more efficient means to analyse them than just traditional means. At present, analysing them is a tedious task as it requires either a large amount of manual processing time and/or highly specialised computational tools. The use of computational tools would speed up this process considerably, however, users almost always do not have access to such tools nor possess the technical expertise to implement or use them. This thesis seeks to explore mechanisms that would assist users without image processing expertise to conduct Video and Image Processing (VIP) tasks in an efficient manner by providing a suitable form of *automated* assistance. For this purpose a combination of computer vision methods, workflow and planning technologies and semantics-based approaches are investigated. These are motivated by the requirements that are outlined in Section 1.3. First the problem overview and thesis aims are presented.

1.1 Problem Domain and Thesis Story

Consider a scenario where video streams are collected continuously (24 hours a day) and saved in formats supported by video processing applications (*e.g.* Avi[1] and Mpeg[2] files). Each video clip is between one and three minutes long. These videos are accessible to users who will traditionally analyse them manually for their needs. To provide a context, consider videos of underwater life available to marine biologists. Among the tasks conducted by the marine biologists are video filtering, object detection and counting. The filtering involves removing videos that are unusable, for instance those that are too dark or too bright as they are uninteresting for further analysis. The detection would include distinguishing objects of interest, such as fish, and further, these objects are counted for statistical purposes. Later on, they may also want to classify the fish according to their species type. Hence there is a range of tasks that the user is interested in. Manual analysis involves observing the video clip, pausing the video to take notes and repeating the process until the task is complete.

The data source used for this thesis comes from the Taiwanese Ecogrid project

[1] A container format that can contain both audio and video; it is one of the most popular formats for video files today.

[2] A universal compression standard format supported by default on most operating systems today.

[33], established and monitored by the National Center for High-Performance Computing (NCHC) and its collaborators. In this project, videos of underwater marine life are collected from various protected national parks and lakes and are made available for research purposes. Such data is valuable for long term monitoring and research especially for marine biologists (domain experts). Studies on fish behaviour, suitable underwater conditions for marine life presence and activity, and population of particular species at a given time are among the scientists' main concerns. However, conducting these analyses manually is extremely time consuming and tedious. Videos collected daily pose an additional constraint on the increasing size of data that need to be processed. What this thesis seeks to do is to provide support in the form of automation to help make this process less cumbersome. This involves investigating the combination of several techniques within an integrated framework. The framework is intended to be interactive to support users' needs to specify preferences and make informed decisions. Fig. 1.1 illustrates a pictorial overview of this scenario.

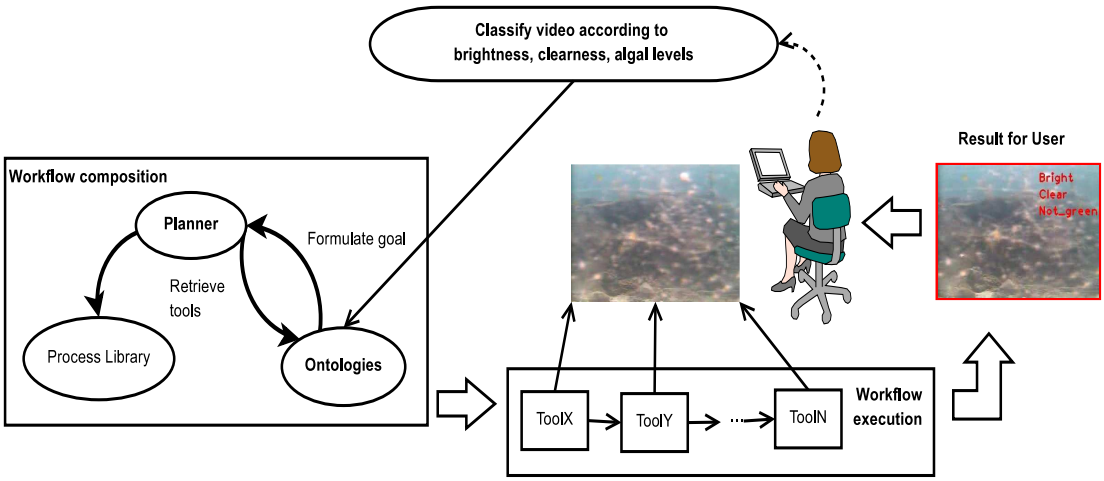


Figure 1.1: Overview of research objective – to assist image processing-naive users conduct video processing tasks using a workflow composition and execution approach.

As can be seen in Fig. 1.1, this cycle involves understanding the user's request, translating the request into machine readable format, then finding a set of tools that can work on the specified data and presenting the result back to the user. For this purpose a workflow composition and execution framework is proposed.

First, existing efforts that provide automated support for analysing large data sets and for generating video processing solutions were reviewed. These include Grid workflows and knowledge-based vision systems. Their features, relevance, contributions and limitations are discussed and a set of gaps still not addressed by state-of-the-

art initiatives are identified. This will be provided in **Chapter 2**.

To address the research gaps within existing workflow and knowledge-based vision systems, a novel hybrid framework for automatic video processing is devised and featured in **Chapter 3**. This framework will make use of two key technologies, ontologies and planning for workflow composition and execution. It will also make use of process models contained in a process library and a set of image processing components.

One aim of this thesis is to help solve a range of video processing tasks that users want to perform. The tasks could also be further specified to include user preferences, such as the processing time for the solution to be computed (fast *versus* slow) and quality of the solution (reliable *versus* robust). The thesis also seeks to integrate, within the framework, well known inferences on the video processing domain to facilitate reasoning and reusability. These capabilities can be provided by ontologies. **Chapter 4** describes the video and image processing ontology constructed for the purpose of this thesis and its usage in the context of workflow composition and execution.

A set of video and image processing components will be required to provide the low level solution steps that will work directly on the videos. **Chapter 5** will explore available computer vision libraries and tailor a set of video and image processing components for use within the proposed framework. These were developed through close collaboration with image processing experts.

Two integral mechanisms for solving the video processing problems are reasoning and execution. For this purpose, planning technology enhanced with workflow execution capability is explored. **Chapter 6** illustrates the workflow enactor and planner in more detail, giving emphasis to the planner's design and algorithm, interaction with the user, and concludes with examples of plan traces for several video processing tasks.

The integrated framework is evaluated on a test set comprising underwater videos of varying quality. **Chapter 7** displays a set of experiments to evaluate the overall system in the aspects of efficiency, adaptability and user learnability. Based on the findings of the experiments, a rigorous analysis of the implemented approach is discussed. Finally, the accomplishments of the thesis are highlighted in **Chapter 8**. The framework's strengths and limitations are discussed. Future directions and reflection based on what has been achieved and what can be improved will also be suggested.

1.2 Research Questions

As a starting point, several scientific questions were formulated for this thesis:

1. What are the requirements for a suitable system that would provide automated assistance for image processing-naive users to conduct a range of typical tasks?

   This question probes the technologies that should be considered to tackle the overarching problem. A further subquestion is how would a combination of these technologies be used within an integrated framework in accordance with these requirements?

2. Ontologies are generally useful for representation and inference in many applications today. What is a suitable form of ontology to represent the relevant concepts and relationships in the video processing problem domain?

   The notion of ontologies gives an added dimension of machine processability for the consensual knowledge captured in a domain. In line with this, how the ontology will be constructed in accordance with ontological engineering practices will need to be investigated. Further to this, the role of the ontology for reasoning should also be exploited.

3. Planning technologies have been successful at solving complex problems with well-defined goals using action sequences. What type of planning approach would be suitable to be used (domain-independent *vs.* domain-dependent, classical *vs.* decomposition-based planning) for generating solutions for VIP tasks?

   The choice to utilise planning techniques for video processing problems is relatively a new one. With regards to this, existing planning techniques are explored and the specification of video processing tasks, domain conditions and solutions as a planning problem are looked into in this thesis. A suitable planning algorithm is also sought for this purpose.

4. Both ontologies and planning technologies have proven to be useful to facilitate representation and inferencing. How therefore would they perform if a combinational approach that uses both of these techniques was implemented? Would this approach be adaptable towards user's preferences?

   This mainly concerns the integration of several techniques and how communication with the user is maintained. Specifically, how is a flexible workflow composition and execution mechanism achieved so as to allow the user to modify their preferences? Is there a mechanism to assist them to make these modifications?

1.3 Requirements

A set of requirements in accordance with the first research question is formulated to address the problem of automatic video analysis for image processing-naive users:

1. Process Automation
   Clearly, some form of automated assistance would reduce the processing time taken by traditional manual processing considerably. In particular, for large sets of data, manual processing alone would be infeasible. The extent of automation used should be investigated.

2. Rich Process Modelling (Iterative Processing, Conditional Branching, *etc.*)
   Videos are made up of sequences of images or frames. Often, analysis would involve some sort of computation over all the frames of a video. At times, different types of computations are performed on a frame due to some conditions, such as user preferences or existing domain descriptions. So, there should be mechanisms to cater for such rich modelling constructs.

3. Performance-Based Image Processing Tool Selection
   In general, many VIP algorithms exist to perform the same family of tasks. For instance, there are many ways in which a model (an image, statistical or algorithm model) to represent the background of a video (called a background model) can be constructed. There should be a mechanism that can select the most optimal background model algorithm for a given video.

4. Adaptable, Flexible and Generic Architecture
   An ideal system would be one that is sensitive to changing user needs. The system should be able to operate on videos of varying quality, such as differing brightness and clearness levels. This is exhibited in videos that are collected continuously in an open environment where the time of day and weather conditions affect the quality of the video, *e.g.* undersea videos. It should also be flexible in that it should be able to process a range of VIP tasks (*e.g.* detection, tracking, classification, *etc.*) with different constraints imposed on them, *e.g.* a solution that is fast but less reliable *versus* a solution that is slow but more reliable.

1.4 Research Contributions

Based on the requirements outlined in the previous section, a hybrid approach that utilises workflow and planning, ontologies and image processing tools would be appropriate to be investigated. Three main claims are made in this thesis.

1. Automated support could be provided for image processing-naive users to perform VIP tasks in a *time-efficient* manner using a novel semantics- and planning-based workflow composition and execution framework. Using this approach is more efficient at solving VIP tasks than using traditional manual methods.

2. Conducting VIP tasks using multiple image processing executables within a planner enhanced with workflow execution capabilities is more *flexible* and *adaptable* towards changing users' needs than constructing solutions using a single image processing executable as employed by typical image processing experts.

3. The planning- and ontology-based automated-assisted mechanism to compose and execute workflows for VIP tasks helps the user *manage* and *learn* the processes involved in constructing optimal solutions. This has not been possible within state-of-the-art workflow and knowledge-based vision efforts.

Summarising the main points from above, a main contribution for the research problem is formulated:

The problem of automatic video and image processing tasks for image processing-naive users can be tackled by the use of a novel workflow composition and execution approach that is achieved by utilising planning and ontologies.

The solution provided by the novel framework takes into account constraint measures such as CPU processing time, amount of memory used, quality of results and initial video descriptions (such as clearness and background movement) in order to perform a performance-based selection. These criteria are formalised in the goal and video description ontologies. Where appropriate, the planner manipulates this information as they are encoded as preconditions within the task hierarchies. The solution plan comprises a set of operators in the form of VIP tools that are formalised in the capability ontology and invoked from a process library. The extensive use of rich process modelling constructs such as iterations and conditional statements allow a high level

of reusability of the VIP tools. In cases where more than one tool is available to perform a task, the user selects a tool with the assistance of the recommended domain descriptions of the tools derived from the capability ontology.

The workflow system is tested on data from an ecological source of varying qualities and environmental conditions. Analysing such data is challenging due to the inherent uncertainties in the user requirements, suitable descriptions for the data and the VIP tools required to conduct the analyses. A context-based/semantics-rich approach is useful to overcome these uncertainties by capturing known user requirements (goals and constraints), video descriptions and VIP software knowledge (capabilities) in ontologies, which are referred to by the planning-enhanced workflow engine which extracts the relevant information required in order to find the solution to perform a VIP task. Thus, this approach enables the rapid development of a system which would normally cost much effort and time to produce. This contributes towards the development of an approximate VIP software that allows non VIP-specialised users to perform video processing tasks more efficiently than they would manually.

Chapter 2

Literature Review

Chapter 1 set the scene for the research problem by highlighting the pervasiveness of video data today that has led to the infeasibility of manual processing by humans to handle this situation. This led to the necessity of providing automated means to conduct video processing, with a particular focus on users without image processing expertise. A set of requirements for a suitable system to overcome this problem was outlined in Section 1.3, which included automated means for workflow composition. In this chapter, initiatives within the Grid workflow and knowledge-based image processing communities were investigated in order to pinpoint the gaps that state-of-the-art efforts have yet to address in the provision of automatic solutions for video processing in line with the requirements. It summarises the findings by outlining the aspects and gaps that will be addressed by this thesis.

2.1 Introduction to Workflow

The Workflow Management Coalition [115] defines a workflow as "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". In other words a workflow consists of all the steps and the orchestration of a set of activities that should be executed in order to deliver an output or achieve a larger and sophisticated goal. A workflow can be seen as a set of activities stored as a model that describes a real world process. Work passes through the model from start to finish, and activities might be executed by people or by system functions. A workflow provides a way of describing the order of execution and dependent relationships between pieces of short-running or long-running work. A workflow enactor or engine manages and

executes models of processes. These models can be created and edited by users who are inexperienced in programming. The flow of information, tasks and interpretation of events are facilitated by the workflow enactor. Among the tangible benefits offered by workflows to an organisation include reduced operating costs, improved productivity and faster processing times [2].

A workflow normally comprises a number of logical steps or functional units, which can be components, tasks, jobs or services. For the purposes of this thesis, they will be referred to as tasks. Tasks are connected in the form of a directed graph and can be primitive or non primitive. Primitive tasks are steps that perform single units of work while non primitive ones manage a set of child tasks. Tasks can also represent logical control structures that define scope and direct the execution flow of the workflow, much as code logic controls, such as `If Then` and `While` loops, control the program flow in code. A task can involve manual interaction with a user or workflow participant, or the task might be executed using machine resources. A task may be performed by the system or by a user. For example, the system may send someone an e-mail message as an alert; or a person might approve a document for distribution.

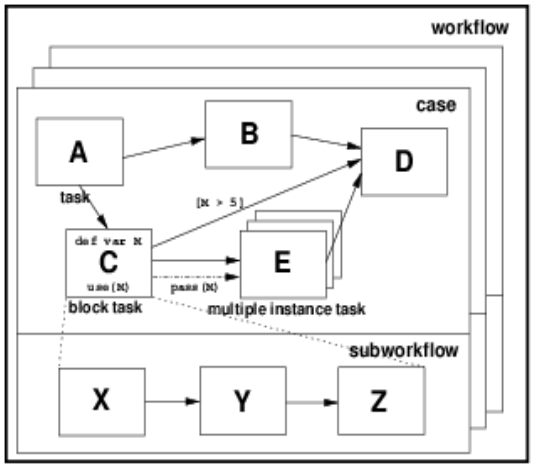


Figure 2.1: Components of a workflow.

Fig. 2.1 [1] shows the components of a workflow and the interdependencies between them. An executing instance of a workflow model is called a process instance or case. There may be multiple cases of a particular workflow model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other. There is usually a unique first task and a

[1] Diagram was retrieved from the Workflow Patterns home page: `http://www.workflowpatterns.com/patterns/data/workflow_structure.php`.

unique final task in a workflow. These are the tasks that are first to run and last to run in a given workflow case. Each invocation of a task is termed a task instance. A task instance may initiate one or several task instances when it completes. This is illustrated by an arrow from the completing task to the task being initiated *e.g.* in Figure 2.1, task instance B is initiated when task instance A completes. This may also occur conditionally and where this is the case, the edge between task instances indicates the condition that must be satisfied for the subsequent task instance to be started *e.g.* task instance D is initiated when task instance C completes if the data element M is greater than 5.

A task corresponds to a single unit of work. Four distinct types of task are denoted: primitive, block, multi-instance and multi-instance block. A primitive task is one which has a simple, self-contained definition (*i.e.* one that is not described in terms of other workflow tasks) and only one instance of the task executes when it is initiated. A block or non primitive task is a complex action which has its implementation described in terms of a sub-workflow. When a block task is started, it passes control to the first task(s) in its corresponding sub-workflow. This sub-workflow executes to completion and at its conclusion, it passes control back to the block task. For example, block task C is defined in terms of the sub-workflow comprising tasks, X, Y and Z.

Workflows may be represented in many forms. As mentioned earlier, workflows are essentially a series of functional units and the dependencies between them define the order in which the units must be executed. Among the well-known models that have been used as the basis for workflow representation languages include Petri nets [41], directed graphs [6], Unified Modelling Language (UML) [14] and Business Process Modelling Notation (BPMN) [116]. In process-aware information systems, including workflow systems, various perspectives can be distinguished [111]. The control-flow perspective captures aspects related to control-flow dependencies between various tasks (*e.g.* parallelism, choice, synchronisation, *etc*). Van Der Aalst *et al.* originally proposed twenty patterns for this perspective [112], but in the latest iteration this has grown to over forty patterns [95]. The data perspective deals with the passing of information, scoping of variables, *etc.*, while the resource perspective deals with resource to task allocation, delegation, etc. Finally the exception handling perspective deals with the various causes of exceptions and the various actions that need to be taken as a result of exceptions occurring.

2.2 Grid Workflow Systems

With the advent of distributed computing in the past two decades, workflows have been deployed in distributed platforms. In a distributed context, such as the Grid or e-Science [35] a workflow can be abstracted as a composite web service, *i.e.* a service that is made up of other services that are orchestrated in order to perform some higher level functionality. The goal of e-Science workflow systems is to provide a specialised programming environment to simplify the programming effort required by scientists to orchestrate a computational science experiment [105]. Therefore, Grid-enabled systems must facilitate the composition of multiple resources, and provide mechanisms for creating and enacting these resources in a distributed manner. This requires means for **composing** and **executing** complex workflows, which has attracted considerable effort especially within the Grid workflow community. A brief overview and analysis of several major workflow systems are provided next. These have been selected due to factors of success and influence on a range of e-Science applications.

2.2.1 Pegasus

Pegasus (Planning for Execution in Grids) [30], is a workflow management system (WMS) which is part of the GriPhyN project [28] that aims to support large-scale data management in a variety of applications such as astronomy, neuroscience, biology, gravitational wave-science and high-energy physics. The WMS is made up of a workflow mapping engine and a workflow executor, Condor's DAGMan [101].

The mapping engine maps abstract workflows to their executable concrete forms. An abstract workflow captures just the computation that the user wants to do at a logical level. The workflow activities are independent of the Grid resources used to execute the activities. The abstract workflow is defined as a directed acyclic graph (DAG) composed of tasks and data dependencies between them. Pegasus requires an input in the specific form of a DAG with XML description (DAX) and produces a concrete (executable) workflow that can be given to the Condor's DAGMan meta-scheduler for execution. In a concrete workflow at the execution level, the activities are bound to specific resources and the necessary data movement to stage data in and out of the computations is included.

The abstract workflows may be defined directly by application developers (workflow experts) according to a predefined schema. They may also be defined semi-automated by using Chimera [36], a Virtual Data System. User-provided partial work-

flow descriptions are fed to Chimera using its Virtual Data Language (VDL). Otherwise abstract workflows may be constructed with the assistance from a workflow editor, such as the Composition Analysis Tool (CAT) [54] which critiques partial workflows composed by users and offers suggestions to fix composition errors and to complete the workflow templates. It assumes that the user may not have the explicit descriptions of the desired goals at the beginning. It utilises classical planning with (component and domain term) ontologies to perform workflow verification.

Once an abstract workflow is constructed, it needs to be mapped to an executable form. This involves finding the resources that are available and that can perform the computations, the data that is used in the workflow, and the necessary software through various Grid information services such as the Globus Replica Location Service (RLS), the Transformation Catalog (TC) and the Globus Monitoring and Discovery Service (MDS). When making resource assignments, Pegasus prefers to schedule the computation where the data already exist, otherwise it makes a random choice or uses a simple scheduling technique. The concrete workflow is produced with a set of submit files necessary for its execution through DAGMan. These files are channelled as submit jobs to Condor-G, a component within Condor. DAGMan is responsible for enforcing the dependencies between the jobs defined in the concrete workflow. Pegasus utilises deferred planning to generate partial executable workflows based on already executed tasks and the currently available resources by a partitioner. This allows for dynamic scheduling that would prevent workflows from failing to execute should any of the resources fail.

More recently Pegasus has been integrated with Wings [40], a workflow creation system that utilises semantic representations of workflows to manage workflows that process large data and computation steps. These are represented using a subset of OWL-DL (description logic-based language) in the form of file and component ontologies. Although this is a step towards performance optimisation and reliability, Pegasus is still limited in that it does not support looping and conditional branching constructs, which are essential for the modelling of iterative processes which are prevalent in video processing. It also requires some level of technical expertise from the user when composing the workflows despite having a mechanism to fix errors and incompleteness in the compositions, *e.g.* via CAT.

2.2.2 Triana

Triana [107, 108] is an open-source project developed at Cardiff University, U.K. It is a problem solving and workflow programming environment that allows users to construct workflows in a graphical manner. It has been used for text, speech and image processing tasks. A user creates a workflow by dragging the desired units from a toolbox and dropping them onto the workspace. Units are interconnected by dragging a cable between them. The resulting composite graph can be executed or saved. At present the Triana custom format (similar to XML) and BPEL4WS [4] are the supported formats. Workflows defined in these formats can also be read and handled by Triana. The GUI allows users to make changes to the workflow by adding, deleting or changing the sequence of execution by drag-and-drop. Additionally, Triana supports looping constructs, which is desirable from a process modelling point of view.

Triana is inherently flow based, it uses both data and control flow to trigger component execution within it. In the case of data flow, data arriving on the input "port" of the component triggers execution and in the case of control flow a control command triggers the execution of the component. Multiple inputs to a component can be set by the component designer to be mandatory, blocking until all received or optional, triggering immediately. The execution of workflow within Triana is decentralised, data or control flow "messages" being sent along communication "pipes" from sender to receiver. The communication can be either synchronous or asynchronous depending on the implementation of the communication pipe.

The internal workflow representation is object based, with specific Java objects for individual component instances or "tasks" and the hierarchy of connected tasks within a network. The representation is a Directed Cyclic Graph (DCG), cyclic connections are allowed within the Triana language, with nodes representing a component and vertices the connections between them. The external representation of the taskgraph is a simple XML syntax. A typical workflow consists of the individual participating component XML specifications and a list of parent/child relationships representing the connections. Hierarchical groupings are allowed with sub-components consisting of a number of assembled components and connections.

Loops and execution branching in Triana are handled by specific components; there is a specific loop component that controls repeated execution over a sub-workflow and a logical component that controls workflow branching. This approach is deemed both simpler and more flexible in that it allows for a finer grained degree of control over

these constructs than can be achieved with a simple XML representation. Explicit support for constraint based loops, such as `while` or an optimisation loop, is often needed in scientific workflows but very difficult to represent. A more complicated programming language style representation would allow this but at the cost of ease of use considerations.

Control units within Triana dynamically rewire the workflow at run-time to connect to the remote services it has discovered based on the distribution policy and the number of services available. A middleware independent abstraction layer, called the Grid Application Prototype (GAP), enables Triana developers to advertise, discover and communicate with Web and P2P Triana Services. The GAP is used to interface with Triana services and provides the middleware independent view of the underlying services and interactions across the Grid. Triana has been a test bed application for GridLab [43], a large EU-funded project and has been integrated with the Pegasus WMS as part of the GriPhyN project.

2.2.3 Taverna

Taverna [88] is an open source workflow engine developed through collaboration between several European academe and industries under the myGrid project [100] and hosted at the School of Computer Science, University of Manchester, U.K. It aims to provide a language and software tools to facilitate easy use of workflow and distributed compute technology for biologists and bioinformaticians who have a deep knowledge of the scientific functionality of the resources they want to link together, but limited expertise in programming and middleware technicalities. It provides graphical interfaces that allows workflow manipulation and workflow progress invocation easily. As it is developed under the myGrid initiative, it makes use of semantic technologies such as RDF [55] and OWL [70] to provide service descriptions that are closer to scientists' view of their experiments than implementation-specific syntactic types.

Taverna adopts a three-tiered data model architecture within an extensible framework for GUI, processor types and external components. The Taverna Workbench acts as the main user interface that allows the construction and editing of workflows. The workflows are written in the Simplified Conceptual Workflow Language (SCUFL) and enacted using the Freefluo workflow enactment engine [38]. The SCUFL language is data centric and is represented using a Workflow Object Model internally to Taverna. Within the execution flow layer, the complexity of the workflow design, which is im-

plicit within SCUFL, is interpreted by a lightweight data model that encodes the data that pass through a workflow. This data model contains some basic data structures such as lists and trees, which bring about an added complexity. Taverna uses an implicit, but configurable, iteration mechanism to handle this. The service interactions to enable the data flow defined by SCUFL are managed by a set of processor plug-ins by presenting a common abstraction over different styles.

Workflow construction is placed in the hands of the domain expert, the scientist. First, the scientist determines the overall intention of the experiment. This informs a top-level design, and would be the overall 'shape' of the workflow, including its inputs and desired outputs. Second, this design is translated into a concrete plan; the choice and configuration of data and analysis services. It provides mechanisms for service discovery and selection through the use of registries and an OWL-based ontology. However, it remains a challenge to construct and maintain an ontology to capture the task information for the bioinformatics domain appropriately and adequately.

Taverna supports fault tolerance through a configurable mechanism; processors will retry a failed service invocation a number of times, often with increasing delays between retry attempts before, finally, reporting failure. This is handled by the Freefluo workflow enactment engine. To provide a more flexible mechanism, Taverna does not attempt to formally structure the domain data. Taverna is particularly suitable for tasks that can handle simultaneous processing as it supports concurrency. However, Taverna does not support automatic composition of workflows. More recently, the task of aiding the composition of Taverna workflows has been provided by Magallanes [94], a web services discovery and workflow composition tool for bioinformatics applications.

2.2.4 Kepler

Kepler [62] is a visual, community-driven project with an extendable open source platform built on Ptolemy II [61], a mature application from the electrical engineering domain that allows users from a range of scientific and engineering disciplines to design and execute scientific workflows. The main project contributors of Kepler include University of California, Davis, University of California, Santa Barbara and University of California, San Diego. It consists of a set of Java packages supporting heterogeneous, concurrent modelling, design, and execution. Users compose workflows using its graphical user interface.

Like Taverna, Kepler is dataflow oriented, with the core description being the pro-

cessing of data through a set of connected actors (processing steps), thus it contains precisely defined models of computation. Kepler's strengths include its mature library of actors, which are mainly local applications, and its suite of directors that provide flexible control strategies for the composition of actors. It is also a modular, activity oriented programming environment that lends itself to the design of reusable components. These components, or processing steps include signal processing, statistical operations, and Boolean logic operations. Kepler can execute processes locally either within the Kepler environment (Java) or within a native environment (compiled native code, or code interpreted by another environment such as Perl).

The Kepler system models a workflow as a composition of independent components that communicate through well-defined interfaces. Workflows within Kepler are serialised in an XML dialect called Modeling Markup Language (MoML). Kepler performs both design-time and run-time type checking on the workflow and data. Processes can be executed in a distributed way, using Web and Grid services. Remotely executed processes behave as a single step in the model of computation regardless of their complexity. Kepler is based on a modular design where different execution models can be easily plugged into the workflows without changing any other components within the workflows. It supports looping and nested constructs. It also has good reliability as it is able to produce partial results even when an entire workflow fails.

Kepler has been used as a knowledge environment for biological and ecological sciences[71] and for workflow provenance[63]. This is exhibited by its strength to model complex computations. However, like Taverna, it also requires the user to possess knowledge in determining the steps involved in solving the task at hand (*i.e.* the workflow construction) and assists them only in the parts of the process that requires software engineering such as programming.

2.3 Workflow Composition Mechanisms

Studies on workflow systems have revealed four aspects of the workflow lifecycle – composition, mapping (onto resources), execution and provenance capture [29]. This thesis will focus on the composition and execution aspects of the workflow lifecycle. Workflow composition can be textual, graphical or semantics-based. Textual workflow editing requires the user to describe the workflow in a particular workflow language such as BPEL [4], SCUFL [87], DAGMan [101] and DAX [30]. This method can be extremely difficult or error-prone even for users who are technically adept with the

workflow language. Graphical renderings of workflows such as those utilised by Triana, Kepler and VisTrails [17] are easy for small sized workflows with fewer than a few dozen tasks. However many e-Science and video processing workflows are more complex. Some workflows have both textual and graphical composition abilities. The CoG Kit's Karajan [114] uses either a scripting language, GridAnt or a simple graphical editor to create workflows.

Some effort in automatic workflow generation has been undertaken in order to ease the tediousness of manual composition. As mentioned in Section 2.2.1, planning technology is used to analyse, verify and correct partial workflows in order to perform interactive workflow composition in CAT [54]. Wings [40] extends this by dealing with the creation and validation of very large scientific workflows. However, CAT requires the user to construct a workflow before interactively verifying it to produce a final workflow. This thesis, in contrast, aims to construct the workflow interactively or automatically. Blythe *et al.* [12] have researched into a planning-based approach to workflow construction and of declarative representations of data shared between several components in the Grid. This approach is extendable to be used in a web services context. Workflows are generated semi-automatically with the integration of the Chimera system [36]. Splunter *et al.* [113] propose a fully automated agent-based mechanism for web service composition and execution using an open matching architecture. In a similar vein to these two approaches, this thesis aims to provide semi-automatic and automatic means for workflow composition, but does not deal with the mapping of resources onto the workflow components.

Three distinct stages are distinguished for workflow creation; the creation of workflow templates, the creation of workflow instances and the creation of executable workflows (done by Pegasus). Workflow templates specify complex analyses sequences while workflow sequences specify data. Workflow templates and instances are semantic objects that are represented in ontologies using OWL-DL. While semantics-based workflow composition is the subject of current research, most efforts have focused on either easing the task of large scale workflows creation for computational workflows and for web services [29]. Next, efforts within the vision community are described, before outlining the research gaps and motivation for this thesis.

2.4 Knowledge-Based Vision Approaches

Several notable efforts have contributed to providing knowledge assisted systems and frameworks for automating the task of video analysis. Since the 1980's **expert systems** played a crucial role in providing knowledge-assisted image analysis. Automated image processing attempts were made by expert systems which generated executable programs from abstract commands. The development of such complex image analysis programs were motivated by the (then) sophisticated Fortran library and limited knowledge representation and reasoning methods. LLVE [67] is a goal-directed image segmentation system which uses image features and transfer processes as fundamental descriptive terms to represent the knowledge about image segmentation. It utilises production rules to guide its search for optimal image processing solutions. CONNY [60] was built to investigate the basic concepts for a self configuring image analysis system aimed at facilitating high flexibility in handling different types of images for different analysis tasks and the direct transfer of human expert knowledge into the knowledge base of the system. It uses a sophisticated evaluation system, unlike previous systems that use a single numerical quality measure for evaluation.

OCAPI [22] attempted to overcome the rigidity of other expert systems by integrating image processing procedures at three levels; physical, syntactical and semantic. It also modelled the relationships between the various entities in the system, making it one of the pioneering systems that attempted at semantic integration that is achieved by ontologies today. COLLAGE/KHOROS [57] is a NASA-driven initiative that aimed at integrating an action-based planner (COLLAGE) to a visual-based library of image processing modules (KHOROS) to aid earth system scientists who study earth's ecosystems. MVP [21] was developed to be used in planetary applications and had the edge over previous systems in that it represented an integration of decomposition and operator-based planning paradigms and used explicit constraints to efficiently reason about operator effects. It utilises hierarchical decomposition planning for search. The BORG system [24] deploys hierarchical-based planning by means of knowledge sources of the Blackboard model that takes into account planning, evaluation and knowledge acquisition issues.

Expert systems were faced with many challenges, namely the lack of mature knowledge representation and reasoning techniques and the difficulty in generalising the image analysis process. The former is now being addressed by machine processable technologies such as ontologies, the latter is still a challenging task for computer vi-

sion experts. In the last decade, the scope for knowledge-based vision efforts has been extended to cope with video data. Also with the advancement in Semantic Web technologies such as OWL [70], RDF [55], RDFS [16] and OWL-S [65], ontologies have been incorporated. Two such initiatives are presented next.

aceMedia [1] is an EU framework project managed by the Informatics and Telematics Institute, Greece with a mix of industrial and academic partners for semantic annotation of multimedia content. Its purpose is integrating knowledge, semantics and content for user centred intelligent media services. The main concept introduced is the Autonomous Content Entity (ACE), which has three layers: content, its associated metadata, and an intelligence layer consisting of distributed functions that enable the content to instantiate itself according to its context (*e.g.* network, user terminal, user preferences). An ACE combines content, metadata, and intelligence into a single entity, allowing automated, knowledge assisted content processing. This initiative has made good use of ontological modelling to allow for semantics-based analyses. These include core, domain and multimedia ontologies described in RDFS [16]. In this respect aceMedia has been a successful effort in utilising ontologies to capture high level descriptions to guide the low level technical tasks of multimedia annotation.

The **Orion Project** [52] addresses the problem of semantic image interpretation by providing a generic and reusable vision platform utilising cognitive faculties. The aspects of cognitive vision involved are reasoning, learning and image processing mechanisms as well as ontology-based representation techniques. They propose a distributed architecture based on three highly specialised modules; a semantic interpretation module, a visual data management module and an image processing module. The platform is used for the detection of plant diseases and for image indexing and retrieval purposes. Two specific works from this project by Hudelot *et al.* [51] and by Maillot *et al.* [64] will be discussed in the next section.

More recent approaches have focused on providing specialised video and multimedia analysis incorporating semantics-based approaches. These include work in video event representation [83], visual ontology for video annotation [48], content-based image retrieval (CBIR) and video event detection [109], Multimedia Understanding through Semantics, Computation and Learning (MUSCLE) Network of Excellence [73] and VIDI-Video project [7]. Nevatia *et al.* [83] developed two formal languages, Video Event Representation Language (VERL) and Video Event Markup Language (VEML) to annotate instances of the events described in VERL. Although this thesis does not propose a new language for describing ontologies, the foundations of VERL

and VEML could be used as a basis when considering the formal notations of the ontologies and rules for inferencing. Hollink *et al.* [48] constructed a visual ontology out of two existing knowledge corpora (WordNet and MPEG-7) by creating links between visual and general concepts in order to use visual information to assist video annotation. This work is also interesting as the visual entities could provide some cues for the video descriptions relevant for this thesis. Town [109] proposes context-dependent inferences given a set of representational or derivational goals using ontologies for image retrieval and video event detection, similar with part of the reasoning proposed by this thesis. The MUSCLE project uses semantics-based and machine learning approaches to provide adaptive and self-learning software tools in the collaborative areas of image and video processing, speech and text analysis, statistics and machine learning. The VIDI-Video project uses similar technologies but to explore the area of semantics-based video search. This thesis, on the contrary, aims to provide a framework that is adaptable to user's changing needs, be it in the task that they want to perform, constraints on these tasks and descriptions of the video. Other commercial and available video analysis tools were not investigated as they essentially work on specific video analysis tasks such as aggregation for video genre classification, object and target tracking in video sequences. These approaches are inflexible and would limit the aims of this thesis that seeks to provide a generic and reusable methodology for general purpose video processing tasks, tools and solutions.

The underlying mechanisms used to compose video and image processing solutions in existing knowledge-based vision systems include rule-based [52], planning-based [24], genetic programming [86] and ontology-based [1, 48, 73]. This thesis proposes a novel approach by combining semantics (ontologies) and decomposition-based planning to construct and execute video and image processing solutions.

2.5 Research Gaps and Motivation

As can be seen from the survey above, automated support for video and image processing has been attempted more by image processing researchers than the Grid workflow community. The next two sections provide comparisons and analysis of the state-of-the-art approaches and what still remains to be done.

2.5.1 Limitations of Current Grid Workflow Solutions

The systems mentioned in section 2.2 possess some similarities and differences that are worth investigating in order to assess their suitability and limitations for the purposes of this thesis. In terms of composition itself, Pegasus differs from Triana, Taverna and Kepler because its main strength is in mapping abstract workflows to their concrete forms, which are then executed by a scheduler. It also provides adaptivity through a partitioner that uses planning to produce partial executable workflows. The role of planning is important for this thesis as it allows for dynamic process selection and composition based on a given set of goals.

Triana, Taverna and Kepler provide a basis for users to create and run scientific workflows. Thus they strive to provide intuitive graphical user interfaces. Both Kepler and Triana provide an interactive visual workflow editor while Taverna only has a static workflow viewer through its custom SCUFL workbench [87]. Triana's interface is one of the most intuitive through its Visual Grid Application Toolkit (GAT), which could be extended by elements of Taverna to support interactive visualisation, but it isn't clear how this could be implemented [106]. Although GUI development is an important aspect for the development of a fully usable system, for the purposes of this thesis, it is not included. However, it will be considered for the development of future prototypes.

Triana, Taverna and Kepler contain similar elements; Triana's tasks are conceptually the same as Taverna's processes and Kepler's actors. The approach in Kepler is very similar to Triana in that the workflow is visually constructed from actors (Java components), which can either be local processes or can invoke remote services such as Web services. In terms of applicability, Pegasus would best suit a domain with well-defined requirements and where the overall goal could be determined from a given set of rules and constraints. Triana is well-suited for composing complex workflows for Web services and Peer to Peer services. Taverna is also suitable to be used in Web and Grid services contexts, but its use may be limited to composing simple workflows, whereas Kepler works very well for composing workflows for complex tasks but it has yet to reach its potential as a fully Grid-enhanced system. Kepler is built upon Ptolemy II which is primarily aimed at modelling concurrent systems. Furthermore, it is designed to be used by scientists which imposes some level of expertise to the user.

While existing workflow systems have more recently incorporated ontologies, their use is still limited. The use of such technologies should not be exclusively independent, rather they should be fully integrated into the system. Existing systems do not provide

full ontological handling nor integration, instead they make use of separate ontology tools to define and manipulate ontologies. The main limitations of existing workflow initiatives can be summarised as follows:

- There is no separation of process and workflow logic, nor the provision of automated support in constructing workflows, thus requiring the user to possess domain expertise.

- They tend to be used in a web services context and do not provide an intuitive overview of workflow systems. It is also unclear as to what execution methods are used and how to deal with failure recovery. Triana and Kepler provide good workflow overview, but are not designed to support 'user-oriented' browsing for producing sets of co-existing customisable (sometimes with conflicted interests) workflows, *e.g.* based on the user's own vocabularies for requirements and goals.

- Unable to improve performance autonomously (or with user involvement) in an incremental manner according to specified goals.

- Generally do not have full integration of ontologies that would allow for more powerful representation and reasoning abilities.

A brief comparison of the four systems indicates that none of them could single-handedly fulfill all the requirements sought for addressing automatic video analysis for image processing-naive users. However, the most prominent and advantageous features exhibited by existing systems were taken into account for devising a suitable framework for this thesis.

2.5.2 Gaps in Knowledge-Based Vision Systems

Most vision-based efforts concentrate on providing highly specialised techniques for very specific application domains due to the high demands for performance and accuracy. Many image processing experts design and develop applications from scratch each time, using trial-and-error cycles and not reusing already developed solutions [92, 93]. Earlier knowledge-based vision efforts cited in Section 2.4 (LLVE, CONNY, OCA- PI, MVP and BORG) were limited to a list of restricted and well known goals. Therefore *a priori* knowledge on the application context (domain-specific concepts such as sensor type, noise, lighting, *etc.*) and on the goal to achieve were implicitly

encoded in the knowledge base. This implicit knowledge restricts the range of application domains for these systems and it is one of the reasons for their failure to be reused for a range of image processing tasks and solutions [32].

Recent approaches [13, 51, 64, 109] bring more explicit modelling but they are all restricted to the modelling of object descriptions for tasks such as detection, segmentation, image retrieval, image annotation or recognition applications. They use ontologies that provide the concepts needed for this description, such as a visual concept ontology for object recognition within Orion [51, 64], a visual descriptor ontology for semantic annotation of images and videos within aceMedia [9] or image processing primitives. Alternatively they capture the knowledge through meetings with the specialists, for instance the use of the NIAM/ORM method in [13] to collect and map the business knowledge to the vision knowledge. However, they do not completely tackle the problem of the application context description (just briefly in [64] and [51]) and the effect of this context on the images (environment, lighting, sensor, image format). Moreover they do not define the means to describe the image content when objects are *a priori* unknown or unusable, for instance in robotics, image retrieval or restoration applications. They also assume that the objectives are well known (to detect, to extract or to recognise an object with a restricted set of constraints) and therefore they do not address their specification.

Thus it could be concluded that there is a lack of modularisation in the way image processing problems are specified within knowledge-based vision systems. The Orion project does not deal with the specification of the vision problem since the goal is fixed to recognition tasks, while some work in aceMedia incorporates several ontologies to describe the domain and image processing aspects but does not tie the goals with the processes involved in solving them. They also use quantitative measures as opposed to qualitative measures for describing their entities. Chapter 4 will illustrate how modularisation and qualitative descriptions could be achieved by the use of several interrelated ontologies within a hybrid framework. To summarise, knowledge-based vision approaches are still limited in the following aspects:

- Lacking modularisation in the way VIP problems are specified. VIP problem tasks and further specifications to constrict the tasks are not expressed explicitly.

- Solving VIP problems using highly specialised hand-crafted solutions in the form of single executable systems targeted at specific tasks, *e.g.* detection, classification, segmentation, *etc.* While this generally aims at higher accuracy, new

solutions will need to be rebuilt from scratch for new data or tasks.

- Ontological efforts are not maximally utilised to encompass aspects such as application context description (*e.g.* lighting and clearness effects) and qualitative measures, which would allow for effective symbolic reasoning.

2.6 Summary

This chapter has shown that while many established and on-going Grid workflow initiatives exist, they have yet to address some vital research gaps, one of which is in the provision of automatic workflow composition which would greatly reduce the engineering and technical efforts required to model workflows. Knowledge-based vision efforts, on the other hand, have recognised the need to provide more user friendly ways to conduct video processing tasks, although they are still very much focused on specific tasks. These efforts are beginning to make use of knowledge engineering technologies such as ontologies, although not to maximal effect which would result in greater flexibility and adaptability. This thesis aims to intersect both fields within a hybrid approach that combines several technologies within an integrated framework. This framework will be described in the next chapter.

Chapter 3

Research Methodology

Chapter 2 discussed related work in the provision of automatic video processing, namely within the Grid workflow and knowledge-based vision communities. The gaps that on-going research have yet to fill were identified, presenting the motivation for this research. This chapter presents a novel hybrid approach that integrates several AI technologies within a rich and flexible framework as a solution. This framework is embedded within three layers containing two key components – ontologies and a planner. These and all other components of the framework are described at a glance in this chapter to provide an overview of the proposed solution. Chapters 4, 5 and 6 will provide detailed workings of the major components and Chapter 7 will demonstrate the effectiveness of the integration in solving several typical video processing tasks.

3.1 Workflow Composition and Execution Framework

A hybrid semantics-based workflow composition method within a three-layered framework was devised and implemented for this thesis. It distinguishes three different levels of abstraction through the design, workflow and processing layers. Each layer contains several key components that interact with one another and with components in other layers. The architecture diagram for this framework is given in Fig. 3.1. The function(s) and components of each layer are described next.

3.1.1 Design Layer

The design layer contains components that describe the image processing tasks, information about the video, image processing tools and processes to be carried out in the

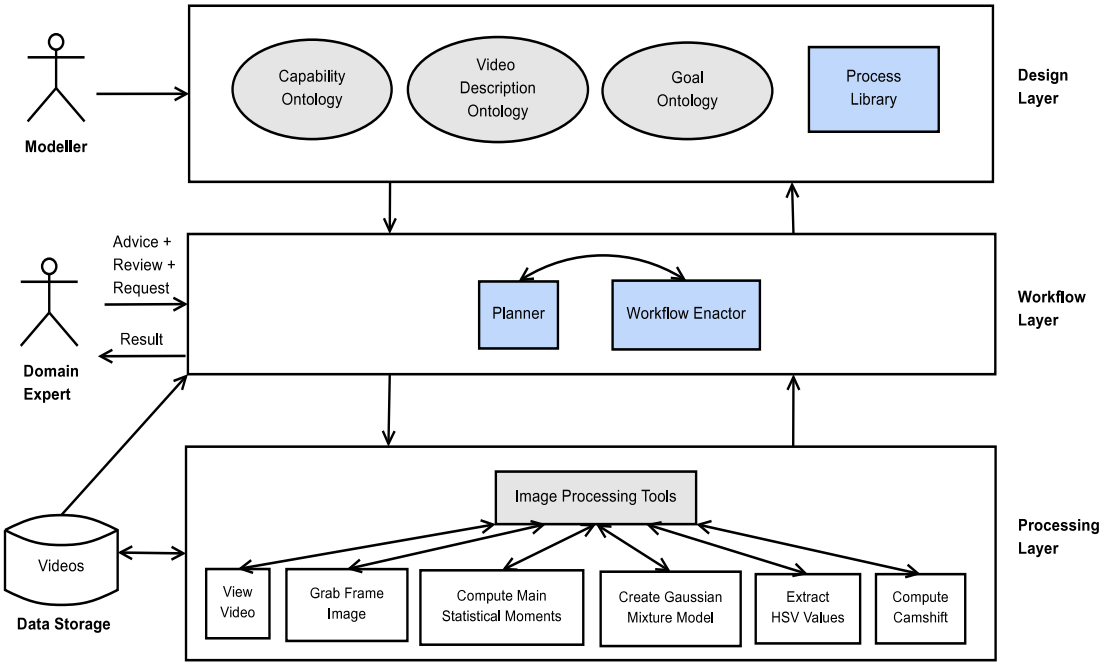


Figure 3.1: Overview of hybrid workflow composition framework for video processing. It provides three levels of abstraction through the design, workflow and processing layers. The core technologies include ontologies and a planner.

system. These are represented using ontologies and a process library. A modeller is someone who is able to manipulate the components of the design layer, for example populate the process library and modify the ontologies. Typically the modeller has training in conceptual modelling and has knowledge in the application domain, but not necessarily. The components could also be updated automatically, as will be shown in Chapter 6. Knowledge about image processing tools, user-defined goals and domain description is organised qualitatively and defined declaratively in this layer, allowing for versatility, rich representation and semantic interpretation.

3.1.1.1 Process Library

A process library is defined as “a collection of code assembled to perform a set of related coordinating and computing tasks” [118]. The process library developed in the design layer of the workflow framework contains the code for the image processing tools and methods available to the system. These are known as the process models. The tools and methodology for their creation are elaborated in Chapter 5. The first attempt in populating the process library involved identifying all primitive tasks for the planner based on the finest level of granularity. A primitive task is one that is not

further decomposable and may be performed directly by one or more image processing tools, for instance a function call to a module within an image processing library, an arithmetic, logical or assignment operation. Each primitive task may take in one or more input values and return one or more output values. For each tool (termed as *independent executable*), its name (user terminology), preconditions, postconditions, input and output are specified. In most cases, the input values are derived from the image processing programs.

In other cases, where these values are instantiated during previous invocations of other independent executables, they are determined during workflow execution. In a similar fashion, the output values of an independent executable could be input parameters to other independent executables. Additionally, the process library contains the decomposition of non primitive tasks or *methods*. This will be explained briefly in Section 3.1.2.2 and at length in Chapter 6. The complete list of independent executables can be found in Appendix A.

3.1.1.2 Ontologies

Ontologies provide formal conceptualisations of entities that are relevant to a particular problem domain so that these representations and the relationships between them are made explicit. Applying such higher level knowledge or semantics would allow for better reasoning on the concepts by the sharing and reuse of existing knowledge and also lead to the discovery of new knowledge.

A video and image processing ontology comprising three sub-ontologies has been incorporated to keep the high level video processing tasks (goals) separate from the low level video and image processing tools (capabilities) and to provide meaning for the entities within a semantically integrated system. Each ontology holds a vocabulary of classes of things that it represents and the relationships between them. Among the possible domain knowledge representations, ontologies present a number of advantages, the most important being that they provide a formal framework for supporting explicit, machine-processable semantics definition, and they enable the derivation of implicit knowledge through automated inference. A system with full ontological integration has several advantages. It allows for cross-checking between ontologies, addition of new concepts into the system and discovery of new knowledge within the system.

The **goal ontology** contains the high level video processing tasks (goals) and constraints that are communicated by the user to the system. Fig. 3.2 contains typical goals or classes of video and image processing tasks. Constraints are criteria that give

additional restrictions to the goal. These include qualifiers to indicate user preferences such as speed of processing, CPU memory used, reliability of result, and accuracy of detection. The goal ontology is discussed in detail in Section 4.4.

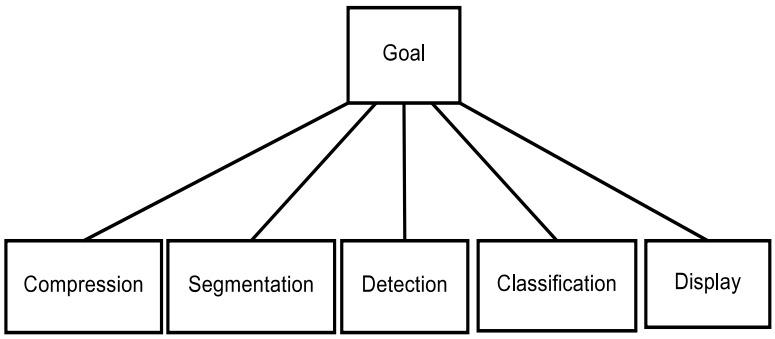


Figure 3.2: Five main types of goals identified for the video processing domain.

The **video description ontology** contains the concepts and relationships that describe the images and videos, such as the lighting conditions, colour information, position, orientation as well as spatial and temporal aspects (see Fig. 3.3 for some examples). Hence, qualitative concepts such as “bright” (high luminosity) and “blur” (low clearness) could be used to describe the input video. The constraints and video description together constitute the domain description. Based on the goal and initial domain information provided by the user, the goal and video description ontologies are used to formulate the input to the planner. The video description ontology is discussed in detail in Section 4.5.

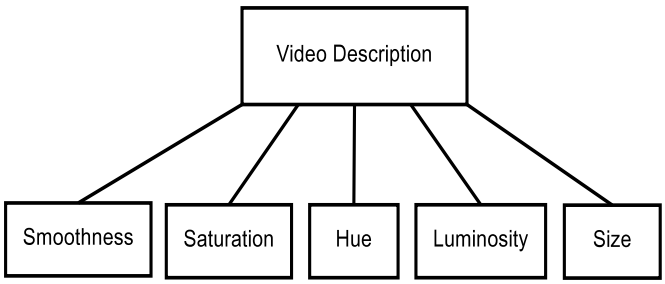


Figure 3.3: Some examples of video descriptions.

As with the process library, the **capability ontology** contains the classes of video and image processing tasks and tools that can perform these tasks. Additionally, it organises them hierarchically, links the tasks to the tools and relates the tools with performance measures. Each task is associated with one or more tools (operators). A tool is a software component that can perform a VIP task independently given some

input values, or a technique within an integrated vision library that may be invoked with given parameters. This ontology will be used directly by the planner in order to identify the tools that will be used to solve the task. The performance level of the tools are also tied to applicable criteria, namely these criteria refer to the domain information (video description and/or constraints). For instance, Create Gaussian Background Model is the best tool to perform a background model creation when the clearness level is high, the speed of movement is high (see Section 6.5.2). The video description ontology is discussed in detail in Section 4.6.

Further details on the ontologies' formalisms can be found in Chapter 4 and Appendix A. An abstract overview of their use in the user-system interaction is provided in section 3.1.2.1 while a concrete example illustrating the derivation of the concepts related to typical image processing tasks is demonstrated in Section 4.8.

3.1.2 Workflow Layer

This layer is the main interface between the user and the system. It also acts as an intermediary between the design and processing layers. It ensures the smooth interaction between the components, access to and from various resources such as raw data, image and video processing toolset, and communication with the user. Its main reasoning component is an execution-enhanced planner that is responsible for transforming the high level user requests into low level video processing solutions. Detailed workings of the planner is contained in Chapter 6.

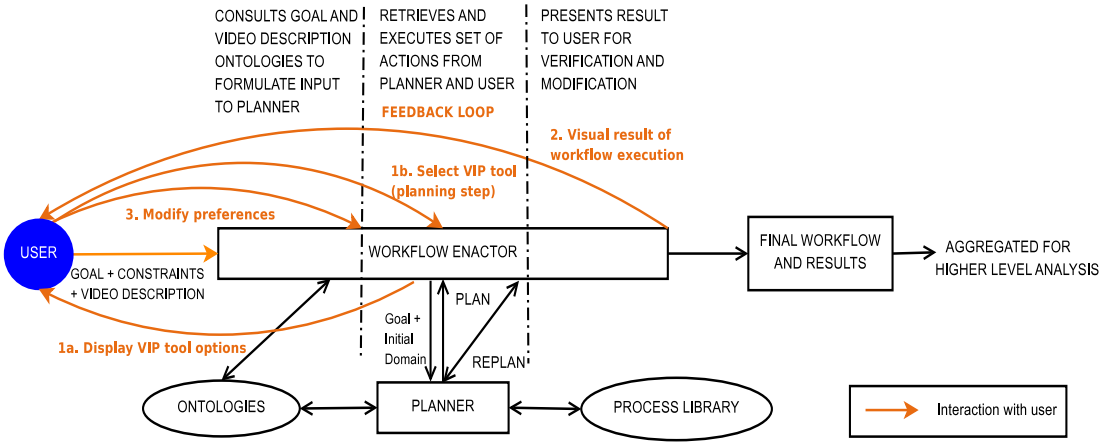


Figure 3.4: Overview of interaction between the user, workflow and other components in the system.

Fig. 3.4 illustrates the flow of communication between the workflow enactor and other components. The communication is initiated by the workflow enactor that interacts with the user for goal formulation. Then it sends this information to the planner, which works closely with the ontologies and process library to retrieve planning steps in the form of video and image processing tools. There are cases when communication is passed back to the user to select video and image processing tools during planning (in semi-automatic mode as shown in steps 1a and 1b). When all these steps have been determined and executed, the visual result is presented to the user by the workflow enactor (step 2). The user may choose to change their preferences and/or request to redo the task with different tool selections (step 3). The result will be presented visually, as before. This cycle of changing preferences, and selecting different planning steps creates a flexible feedback mechanism between the user and the system.

3.1.2.1 Workflow Enactor

The workflow enactor plays the important role of choreographing the flow of processing within the system. It should be noted that unlike the workflow enactors covered in Chapter 2, the workflow enactor developed for this thesis does not deal with resource allocation and scheduling, rather, on the composition of specific operators and the execution of the operators given their predetermined parameters. First it reads in the user request in textual form (use selects from a list of options). Next it consults the goal and video description ontologies to formulate the input that is then fed to the planner. When the planner, with the assistance of the process library and capability ontology, returns the final solution plan, the enactor prompts the user for further action. The user has access to the final result of the video processing task textually and visually, has the choice to rerun the same task on the same video but with modifications to the domain information, rate the quality of the result or perform another task. The composed workflow is saved in a script file that can be invoked easily off-line. By being able to view the result of each solution with changes to the domain information, the user can assess the quality of the solution produced. This feedback mechanism could be used as a basis for improving the overall performance of the system as verifying the quality of the video processing solutions automatically is not a trivial task. The workflow enactor is described in more detail in Section 6.2.

3.1.2.2 Planner

The planner acts as the "brain" of the system, which translates the high level user request into low level image processing steps. Adopting the principle that image processing tasks can be solved using a hierarchical decomposition approach, a Hierarchical Task Network (HTN) planner [96, 103] was implemented to realise this. This principle states that a task (goal) may be achieved by performing a set of primitive or non primitive subtasks, each non primitive subtask is further decomposed recursively until primitive tasks are reached. As stated earlier, a primitive task could be performed directly by an image processing tool (operator). The role of the planning algorithm is then to select the optimal set of image processing tools to achieve a given image processing task. In classical planning, the goal and initial state (domain information) are given as input and the planning algorithm would have to consider a large search space before finding the best plan to achieve that goal. In HTN planning, in addition to the goal and initial state, a set of *methods* are provided to the system. These methods encode the decomposition of known tasks. For example, a simple video classification task may be achieved by first preprocessing the video, followed by computing the average values for the attributes to be classified. Computing the average values for the attributes involves computing the brightness, clearness and green tone levels in each frame image accumulatively. These best known practices adopted by image processing experts or heuristics are included as methods in the process library (Section 3.1.1.1).

In an HTN planner, the search space is reduced greatly because only the subtasks that are applicable to solve a current task are considered as nodes for further expansion. The set of options are reduced as planning progresses as only those options that match the preconditions for a subtask (either primitive processes or methods) are selected as valid choices. HTN planners are more efficient as a result of this. The planner, which works closely with the process library and capability ontology, passes the plan generated to the workflow enactor which presents the solution to the user. In cases where the user chooses to modify the domain information, the planner would replan according to these modifications. The full workings of the planner is provided in Section 6.3.

3.1.3 Processing Layer

The processing layer consists of a set of video and image processing tools that can perform various image processing functions. The functions of these tools are represented in the capability ontology in the design layer. Once a tool has been selected by

the planner, it is applied to the video directly. The final result is passed back to the workflow layer for output and evaluation.

Image Processing Toolset. The set of video and image processing tools available for performing various image processing operations are generated using OpenCV [53], an extensive open source computer vision library. A few tools that are shown in Fig. 3.1; View Video, Grab Frame Image, Compute Main Statistical Moments, Compute Gaussian Mixture Model, Extract HSV Values and Compute Camshift. The functions (primitive tasks) that they can perform are represented semantically in the capability ontology, described in Section 4.6. It should be noted that there could be more than one tool available to perform a particular function. For instance, there are seven tools that can perform the task "create background model". A list of all the video and image processing tools available can be found in Section 5.5 and Appendix A. As these tools were developed through close collaboration with image processing experts, the methodology of their design and construction is described in Chapter 5.

3.2 Summary

This chapter has outlined the workflow composition and execution framework proposed by this thesis. It consists of three layers of abstraction; design, workflow and processing. Each layer contains several key components that provide the function(s) of that layer and are loosely integrated with other components. The two core technologies of this framework, ontologies and a planner, were also highlighted. This integrated approach has been implemented and evaluated on underwater videos. This overall methodology for problem solving is generalisable for domains other than video processing, however, this has not been validated. Chapter 7's conclusion will touch on aspects of genericity of this framework within the video processing domain using data sets other than underwater videos. The next three chapters provide the technical details of the video and image processing ontology (Chapter 4), the video and image processing tools (Chapter 5), the planner and workflow enactor (Chapter 6).

Chapter 4

Video and Image Processing Ontology

Chapter 2 argued that the main challenges for automating the steps involved in video and image processing tasks over the past few decades lie in the fact that existing knowledge representation and reasoning methods lack maturity and the image processing formulation is hard to generalise. This chapter outlines how ontologies could be used for representation and inference within the video processing field. A modular video and image processing (VIP) ontology consisting of three sub-ontologies – goal, video description and capability – was constructed and incorporated into the workflow composition and execution framework. The sub-ontologies were briefly introduced in Chapter 3. In this chapter they are examined in further detail, giving emphasis on the methodology of their construction via modularisation and how they are used to capture *succinctly* the three main aspects for the video and image processing domain for this thesis. The key innovations of the methodology include the reuse and refinement of existing ontologies, as well as the introduction of new aspects in a collaborative manner, adhering to the guidelines laid out by sound and established ontological engineering practices. Their roles in inference include guiding the workflow according to user requirements, consistency checking and assisting the naive user make decisions at certain points of execution. These roles are demonstrated in Section 4.7. The chapter concludes with a walkthrough of ontology use for a detection task.

4.1 State-of-the-art

The field of ontological engineering [5, 42] has gained much attention in the past decade. Ontologies are used for capturing knowledge and semantics in a domain and have been used widely in several major fields including medical, linguistic and enter-

prise. Domain ontologies are often modelled in a collaborative effort between domain and ontology experts to capture *consensual* knowledge that is formed between the domain experts that can be shared and reused among them. In the video processing field, ontologies are extremely suitable to many problems that require prior knowledge to be modelled and utilised in both a descriptive and prescriptive capacity since they encode the concepts and relationships between the components in the world. Several major efforts including those by expert systems, projects such as VIDI-Video [7], Orion [52], aceMedia [1] and Pantheon [24] were described in Chapter 2. Some advances within the development of video and image processing ontologies were pertinent in recent years. As part of a Challenge Project on Video Event Taxonomy, Nevatia *et al.* [83] developed a formal language for describing an ontology of events, Video Events Representation Language (VERL) and Video Events Markup Language (VEML). This was based on a collaborative effort between computer vision and knowledge representation and reasoning communities. Its focus was to model video events as composable tasks made up of simpler subtasks and primitives and has been used for annotating video events [37]. Work by Town [109] has focused on using ontologies to guide content based image retrieval (CBIR) and video event detection. Another effort by Colantonio *et al.* [25] has built an image processing ontology based on an existing image processing thesaurus. Project VIDI-Video and aceMedia have worked on developing multimedia ontologies, again for some specific tasks (such as annotation). The key issue is that each effort has focused on providing suitable ontologies for the particular video and image processing problem(s) that they were trying to address. None of them worked on describing the VIP field in *generic* terms.

The Networked Ontologies (NeOn) [81] project is a four-year E.C. project involving 14 European partners led by the Knowledge Media Institute at the Open University, U.K. It aims to create a methodology for building ontology networks that provide guidance on all the key aspects of the ontology engineering process, including collaborative ontology development, the reuse of ontological and non-ontological resources, and the evolution and maintenance of networked ontologies. This is encapsulated in the forthcoming book “NeOn Methodology for Building Ontology Networks” [82] to be published as an end product of the project. While not directly related to the thesis aims, one aspect of interest in the NeOn project is that it has developed an ontology on the fisheries domain that could be considered to be used in the future.

An interesting work by Renouf *et al.* [93] attempted to represent the VIP field generically by studying the formulation of image processing applications in order to

propose an ontology that is used to express the objective of the domain expert (the **goal** part of the ontology) and define the image class to be processed (the input images and their variability description using the **domain** part of the ontology). It is used in the Hermes project [90] which proposes a human-machine interface dedicated to domain experts inexperienced in the VIP field. Using this interface, users are able to formulate their goals and the description of their images using their domain knowledge.

As a result, the Hermes project built an image processing ontology that describes image processing concepts in a comprehensive manner without restricting the image processing task to just detection or classification and so on. The ontology was divided to describe different aspects of the formulation of an image processing problem; system, objective and image class. The system component models an image processing application as an intermediate system that consumes images from an image producer and generates images for a post-processing system. The objective component models the list of image processing goals into six possible tasks; restoration, enhancement, compression, reconstruction, detection and segmentation. Apart from these, constraints that constrict the objectives were also modelled in the ontology. The type of constraints include regulation constraints, feedback constraints and control constraints. Regulation and feedback constraints can be viewed as functional requirements, while control constraints can be viewed as non-functional requirements in software engineering terms. The image class definition then models the description of the input image itself, such as noise, spatial relation, size, motion, colour, brightness, blurness, shape and texture features, among others. A visual representation and documentation of the image processing ontology can be obtained in [91].

Considering that the image processing ontology constructed by the Hermes project is comprehensive, captures the aspects of VIP domain succinctly and has the ability to describe more general video and image processing tasks as compared to other efforts, it was reused, modified and refined to be used in a planning enhanced with workflow capability context. How this is done and achieved are described in the next section.

4.2 Modularisation Methodology

For the purposes of this research, a set of ontologies was required to model the video and image processing (VIP) field so that it can be used for domain description and understanding, as well as inference. The ontology should describe the domain knowledge and support reasoning tasks, while being reasonably independent from the system. The

principles adopted for the ontology construction included simplicity, conciseness and appropriate categorisation. For this reason, several aspects of the VIP field were highlighted. These were identified as *goal*, *video description* and *capability*. These aspects were motivated by the context of their use within a planning system that requires the goal and initial domain state model (which includes the initial video description) and also a performance-based selection of operators. It was noted that the objective and image class components of the Hermes image processing ontology approximately correspond to the goal and video description aspects. The capability aspect would model the VIP capabilities, such as matching tools for solution functions, as well as their suitability with respect to numerical and non numerical criteria to perform their respective functions. This was not provided by the Hermes ontology.

Following the SUMO (Suggested Upper Merged Ontology)[1] ontology representation, a modular ontology construction was adopted (see Fig. 4.1). The modularisation aims to separate the formulation of the problems from the description of the data and the solutions to be produced.

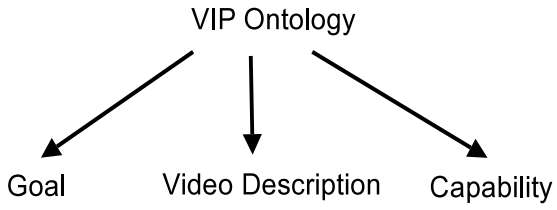


Figure 4.1: Modular structure of the Video and Image Processing (VIP) Ontology.

The VIP ontology consisted of three modular sub-ontologies that could capture the main aspects in a seamless manner. The existing Hermes ontology was split into two sub-ontologies and an additional ontology was incorporated into the VIP ontology. The two ontologies derived from the Hermes ontology were modified accordingly. Details of the modifications will be provided in sections 4.4, 4.5 and 4.6. The resulting three sub-ontologies based on a collaborated effort with knowledge-based vision experts is presented in [78].

Although the Hermes ontology provided a comprehensive representation of the image processing field, it lacked rich semantics such as relationships between the concepts which meant that it was not ideal for reasoning. There were limited levels of expressiveness between the concepts in the ontology apart from the subclass *is-a* relation, making it an informal *is-a* ontology. The next stage in the construction phase involved

[1] http://www.ontologyportal.org/

defining more relations to express richer relationships between the components of the ontology. The ontology representation and components are described next.

4.3 Representation

A suitable declarative language for representing the VIP ontology was required. Several formalisms were considered, including Web Ontology Language (OWL) [70], Business Process Execution Language (BPEL) [4] and Semantic Web Rule Language (SWRL) [49]. OWL, although an emerging standard for the Semantic Web, is not suitable to be used in a workflow execution context. One of the aims of the ontology representation language is that it should support workflow enactment for execution purposes, consistent with the integrated framework outlined in Chapter 3. BPEL, on the other hand is a prominent execution language, however, its use is applicable for web services execution. SWRL is heavily rule-based and is not rich enough for expressing all the features required for the ontology. As will be shown in Chapters 5 and 6, expressing VIP tasks in a planning context involves much process modelling. Hence, a language that suited all these requirements, was selected and will be described next.

4.3.1 Fundamental Business Process Modelling Language

Fundamental Business Process Modelling Language (FBPML) [20] is a merging and adaptation of two recognised process modelling languages: Process Specification Language (PSL) [97] and Integrated DEFinition for Process Description Capture Method (IDEF3) [68]. FBPML combines the formal semantics provided by PSL complemented with the rich visual and modelling methods from IDEF3, so that formal analysis and reasoning may be carried out. Therefore, FBPML is both visual and formal and can be used to support workflow execution via its declarative syntax. It has both data and process modelling capabilities. In FBPML, the term process, activity and task are used interchangeably. A model described in FBPML is made up of ‘Main Nodes’, ‘Junctions’, ‘Links’ and ‘Annotations’. Some of the process elements of FBPML are shown in Fig. 4.2 and will be described next.

Main Nodes: ‘Activity’ is the main concept to denote a process which may be further decomposed or specialised into subprocesses. The three main components of an activity are ‘Trigger(s)’, ‘Preconditions’ and ‘Action(s)’. ‘Primitive Activity’ is a leaf node activity that may not be further decomposed or specialised. Primitive activities

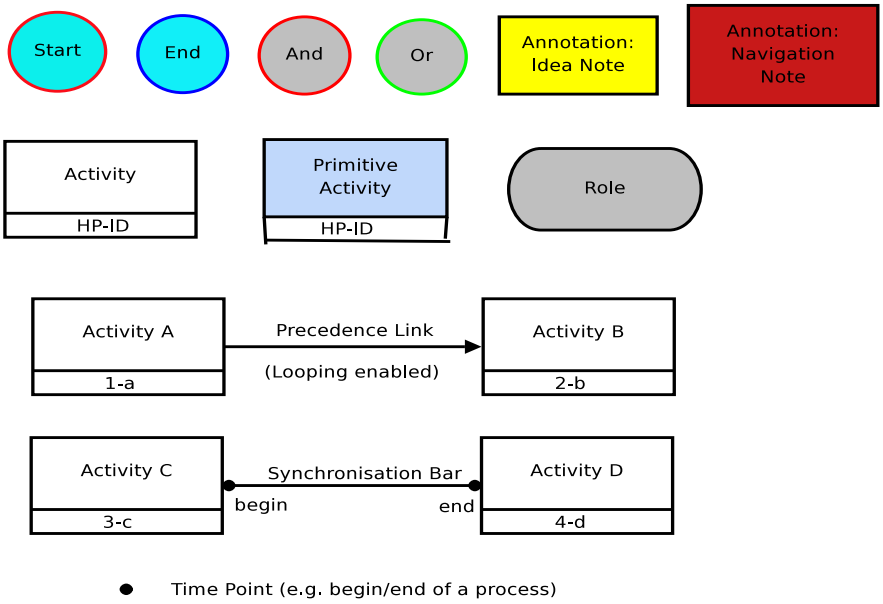


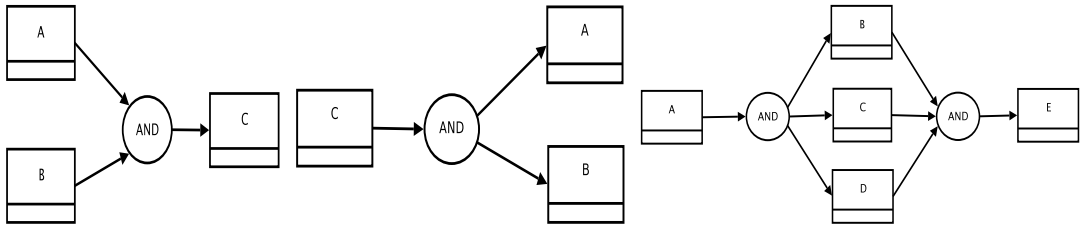
Figure 4.2: Main components of FBPML.

are directly connected to application layers. ‘Role’ is a specialised function, capability or authority possessed by an enabler (an individual, a group of people or a software component) over a set of activities, that is a responsibility in context. ‘Time Point’ is used to express temporality and indicates a particular point in time during the process execution. Graphically time points are represented as the circular ends of either sides of a ‘Synchronisation Bar’.

Links: Links between processes consist of ‘Precedence Links’ and ‘Synchronisation Bars’ which place temporal constraints on process execution. ‘Precedence Link’ is comparable to the more constrained Precedence Link, type II, in IDEF3. It indicates that the latter activity cannot start until the former has finished. ‘Synchronisation Bar’ also places a temporal constraint between two time points. This notation enables any time points to be made equivalent and therefore enables process operations to be synchronised or executed concurrently.

Junctions: Junctions are used to connect multiple activities. They also define the temporal constraints and control the initiation and finishing of parallel processes. The four types of Junctions in FBPML are ‘Start’, ‘End’, ‘And’ and ‘Or’. ‘Start’ and ‘End’ denote the commencement and completion of a process model execution. Each use of a junction is a type of one-to-many (Split junction) or many-to-one (Joint junction) relationships. The two most commonly used junctions in the split and joint contexts are ‘And’ and ‘Or’. A subset of the ‘Or’ construct is the ‘Xor’, which imposes that only

one process is selected for execution. Their semantics are equivalent to the logical connectives AND, OR and XOR, respectively (Fig. 4.3).



(a) And-Joint. (b) And-Split. (c) And-Split-Joint.

Figure 4.3: Main 'And' junctions in FBPML.

An And-Joint indicates that there is more than one process preceding the And junction but there is only one process following the junction. In Fig. 4.3(a), both A and B must complete execution before C can begin. In an Or-Joint, one or both of the processes can be executing, in which case all the executing processes must complete before the process following the junction can begin. In an Xor-Joint, only one process will precede the Xor junction and one process following the junction. Semantically, an And-Joint indicates the process execution flow and the temporal constraint that all of the preceding processes must be finished before the following process is temporally qualified and therefore be executed.

And-Split or Or-Split means that only one process will proceed to the And or Or junction, but more than one process will follow the junction. An And- or Or-Split indicates that all of the following processes become temporally qualified when the preceding process is finished. Furthermore, an And-Split also indicates that all of the following processes must be executed at some point of time after the preceding process is finished. In Fig. 4.3(b) C must finish executing before A and B can start. In an Or-Split context, either A or B or A and B can be selected for execution. Fig. 4.3(c) shows a combination of And-Split and And-Joint to produce a set of processes that must start at the same time point after the preceding process and also must all finish before the following process can start. This can be applied to an Or-Split-Joint to indicate a set of one or more processes and an Xor-Split-Joint to indicate only one process execution. Other combinations of junctions are also possible to represent more complicated models. Some process models will be illustrated in Section 5.6 for the modelling of VIP tasks.

Annotations: Annotations include 'Idea Note' and 'Navigation Note'. Neither of

them contribute to the formal semantics of the process model. Instead, they are used to help users to understand the processes more clearly from an intuitive point of view. 'Idea Note' records information which is related to the processes, while 'Navigation Note' records the relationships between diagrams in a model.

The data language, FBPML Data Language (FBPML-DL) is first ordered and consists of foundational model, core data language, extension data language and meta-predicates. The foundational model encodes concepts, predicates and functions of background theories that are used in the language such as datatype definitions, logical operators, quantification operators and constants. It also introduces primitive predicates that will be used to define other predicates. The fundamental and common predicates used by all applications are defined in the core data language. The extension data language includes predicates and functions that are additional to the core data language that has been provided by FBPML. They are usually defined by the user and are often application and domain-dependent. Meta-predicates give definitions for other predicates and may define axioms of an application model. The core data language and meta-predicates are of relevance for the purposes of the VIP ontology representation.

Its syntax follows the convention provided by Prolog. FBPML can be translated to OWL [75] using data model (representation) and process model (execution) translations. Although the process model translation is limited, the data model translation is more direct and hence ontologies represented using OWL can be translated to FBPML-DL and used in an execution context.

4.3.2 Main Concepts and Relations

As mentioned above, the syntax of FBPML follows the convention provided by Prolog. Hence, the concepts and relations are described in this syntax, where appropriate. A unary predicate class/1 is used for representing concepts in the ontology. N-ary functions are represented using n-ary predicates. The main ones include the following:

- subclass_of/2 to represent a class-to-class relationship using a is-a concept. A few examples will be given in each ontology.
- instance_of/2 to represent individuals of a class, it is an instance-to-class relationship. A few examples will be given in Section 4.4.
- class_rel/3 to represent binary relationships between two classes. A few examples will be given in each ontology.

- specialisation_of/2 is and instance-to-instance relationship to represent a more specialised instance than a root instance. It is used in the capability ontology. This will be elaborated in section 4.6.

- instance_att_list/2 represents a property of an instance given in a list. A few examples will be given in the capability ontology.

- canPerform/2 is an instance-to-class relationship within the capability ontology. This will be elaborated in Section 4.6.

- hasDescription/2 provides a textual description (String) of an instance. The description is a user friendly definition of that instance, *e.g.* hasDescription(create_background_model, 'creates an image that represents the background so that objects on the current frame image can be detected.').

- hasPerformanceIndicator/2 provides a textual description (String) of the most suitable domain conditions for an instance. The description is a user friendly description that aids the user to make an informed selection of tools in semi-automatic mode.

4.3.3 Functions and Axioms

Functions are special cases of relations where the *n*-th element of the relation is unique for the *n-1* preceding elements. One function that is useful for VIP tasks is a conversion from qualitative to quantitative values using a conversion function. Axioms serve to model conditions that are always true, normally used to represent knowledge that cannot be formally defined by the other components. In addition, formal axioms are used to verify the consistency of the ontology itself. Some axioms have been formulated for the VIP domain. Their definitions (and logical representations where appropriate) are provided below:

Axiom 1. A video is an ordered set of images. Therefore, any goal (task) applicable to an image is also applicable to a video (but not the other way around).
applicable(goal(X), data_type(video)) :- applicable(goal(X), data_type(image)).

Axiom 2. No colour functions can be applied to a greyscale image, *e.g.* identify blue fish in a greyscale image. This is due to the differences in their value and spatial

trajectories. A colour image is represented in 3-dimensions in value and 2-dimensions in spatial trajectories respectively. A greyscale image is represented in 1-dimension in value and 2-dimensions in spatial trajectories.

Axiom 3. An object in an image has three main features – colour, shape and texture. Two objects with similar features can be concluded to be similar.

Axiom 4. Domain independent axiom. X is an instance of a class Y if it is an instance_of Y or if X is a specialisation_of another instance Z which is an instance of Y.
instance(X, Y) :- instance_of(X, Y).
instance(X, Y) :- specialisation_of(X, Z), instance(Z, Y).

Axiom 5. Domain independent axiom. A class C is a descendant of a class A if it is a subclass of A or if it is a subclass of another class B, and B is a descendant of A. This is encapsulated in the predicate descendant_of/2.
descendant_of(C, A) :- subclass_of(C, A).
descendant_of(C, A) :- subclass_of(C, B), descendant_of(B, A).

Axiom 6. If an instance I has type X, then it cannot be of type Y, where X and Y are not equal and are from the same hierarchy, *i.e.* X and Y are descendants of the same parent class. This axiom could be used to detect type mismatches.
type_mismatch(I, X, Y) :- instance_of(I, X), instance_of(I, Y),
X =\= Y, descendant_of(X, Z), descendant_of(Y, Z).

There are other axioms that are informative but not part of the research questions of this thesis, these include axioms for a relational algebra, reflexivity, irreflexivity, symmetry, asymmetry, antisymmetry, transitivity and composition of relations, inverse relations, (exhaustive) partitions, axioms for subrelation relationships, axioms for part-whole reasoning, nonmonotonicity and axioms for temporal and modal contexts. Alvarez *et al.* [3] provide axioms specific to image processing such as architectural axioms to define causality/pyramidal properties, comparison principle and morphological axioms. The next three sections describe the sub-ontologies in more detail.

4.4 Goal Ontology

The goal ontology contains the high level goals and constraints that the user will communicate to the system. These are represented by the concepts `Goal`, `Constraint`

`Category`, `Constraint Descriptor` and `Constraint Qualifier` as illustrated in Fig. 4.4. They were adapted from the Hermes ontology.

The straight arrows denote subclass or *is-a* relations and the dotted arrows denote other relations with the name of the relations specified. Instances appear as texts under the boxes. Under the class `Goal`, the main goals are defined to be `Compression`, `Segmentation`, `Restoration`, `Enhancement`, `Classification`, `Detection` and `Display`. In the Hermes ontology, detection and classification are not included under the goal umbrella but have been specified as a post-processing objective (under system model). As the system model is not included in the VIP ontology, these concepts were moved to the goal ontology. `Constraint Category` and `Constraint Descriptor` refer to the conditions that restrict the video and image processing tasks or goals further. The main constraints are highlighted in Fig. 4.4.

`Performance Criteria` allows the user to state whether the goal that they wish to perform should be executed using a faster algorithm (indicated by the criterion `processing time`) or whether it should take less (CPU) `memory`. `Quality Criteria` with the value `reliability` constrains the solution to be the most accurate result. If such a solution could not be found, then the system should fail rather than produce alternative options. `Robustness` indicates the reverse; that the system should not break down completely in cases where a reliable solution could not be found, instead it should return an alternative (imperfect) result. Also incorporated are the criteria for `Accuracy` – `prefer miss than false alarm` and `prefer false alarm than miss`. Miss and false alarm are terminologies used within VIP tasks that involve the detection of objects to indicate the accuracy level of the detection. Consider a real object to be the object that needs to be detected. A miss (false negative) occurs when a real object exists but is not detected. A false alarm (false positive) occurs when an object that is not a real object has been detected. The criteria for `Occurrence` is used for detection tasks to constrict the number of objects to be detected. `All occurrences` imposes that all the objects should be identified while `at least one occurrence` does not. The binary relation, 'is_related_to', is defined to tie the constraints to the related goals. These constraints are only applicable to the class that they are related to and will not be required for other goals during goal formulation. This is defined using the class relation, class_rel/3 predicate:

```
class_rel(is_related_to, accuracy, detection).
class_rel(is_related_to, occurrence, detection).
```

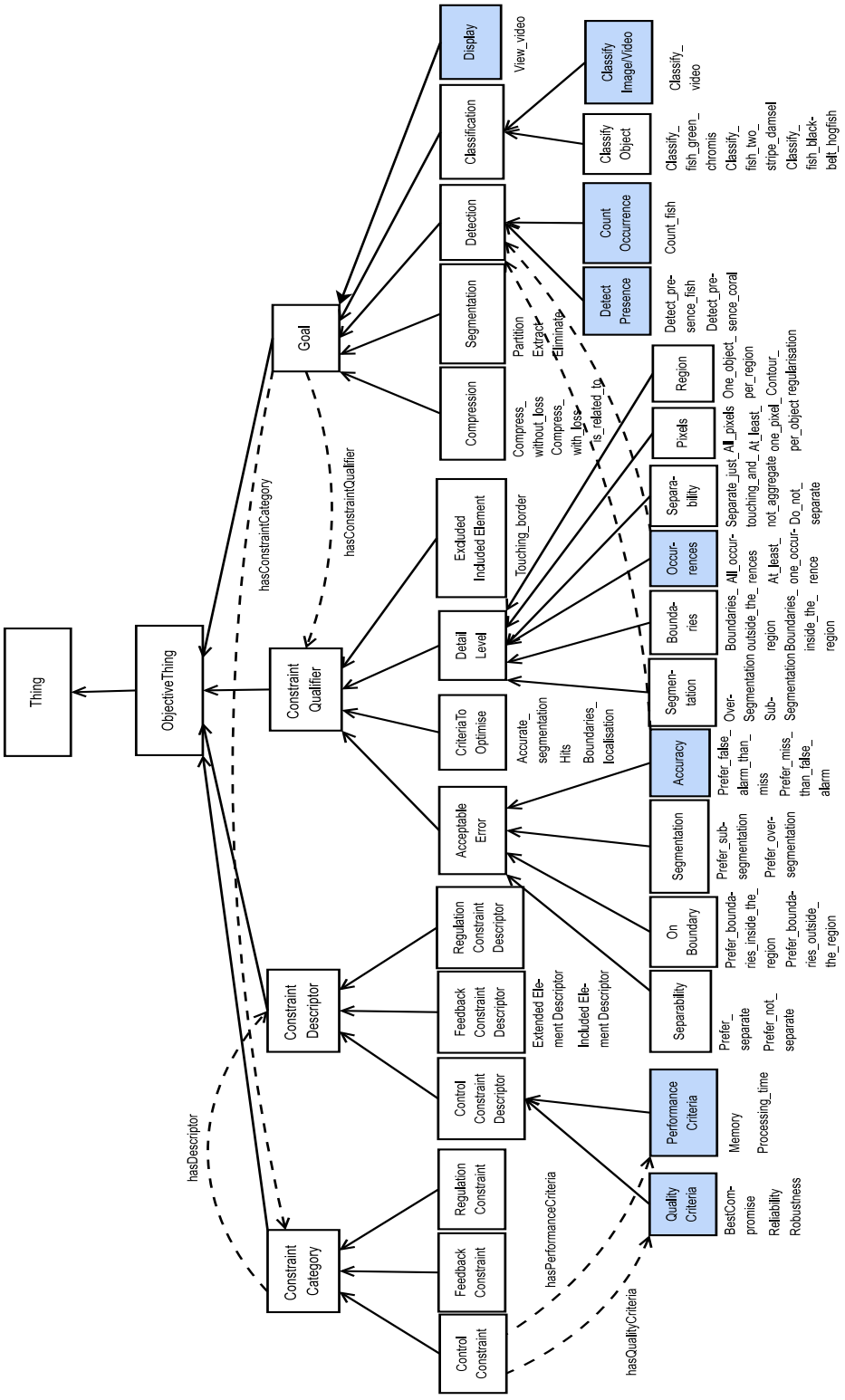


Figure 4.4: Goal Ontology at a glance. The concepts in blue denote the main classes of goals and constraints that were applied to the examples in this thesis.

These two facts indicate that the constraints `Accuracy` and `Occurrence` are related to the goal `Detection`. The goal ontology is mainly used for representation and to assist with goal and constraints selection for the user. This will be illustrated in section 4.7.1.

4.5 Video Description Ontology

The video description ontology describes the concepts and relationships of the video and image data, such as what constitutes video/image data, the acquisition conditions such as lighting conditions, colour information, texture as well as the range and type of their values and spatial relations. The user will have the choice to specify video descriptions and/or the constraints after specifying the goal. As explained in Chapter 3, the system will use the goal and video description ontology to build the user request before feeding it into the planner which will be responsible for the solution generation.

Fig. 4.5 gives a pictorial overview of the main components of the video description ontology. The main classes include `Video/Image Class Definition`, `Descriptor` for the `Video/Image Class`, `Descriptor Value`, `Relation` and `Measurement Unit`. The `Video/Image Class` contains concepts that relate to what could be considered as VIP entities, such as *image, video, object, edge* and *region*, described as `Visual Primitives`. Apart from the main concepts, the `Acquisition Effect` also define the video/image class. These include the colour, noise, clearness (smoothness) and movement effects contained in the video/image. The descriptor for the visual primitives and acquisition effects are contained under the `Descriptor` class and are connected to the `Description Element` via the relation 'hasDescriptionElement'. `Visual Primitive Descriptor` describes the video/image class such as its geometric and shape features, *e.g. size, position* and *orientation* while `Acquisition Effect Descriptor` contains the effects of the whole video/image that contains the video/image class such as the *brightness (luminosity), hue* and *noise* conditions. The values that these descriptors can hold are specified in `Descriptor Value` and connected by the class relation 'hasValue'. For the most part, qualitative values such as *low, medium* and *high* are preferred to quantitative ones (*e.g. numerical* values). Qualitative values could be transformed to quantitative values using the 'convertTo' function. This would require the specific measurement unit derived from one of the classes under the concept `Measurement Unit` and conversion function for the respective descriptor. *e.g.* a low velocity could be interpreted as movement with velocity within a range of 0 and $25\text{ms}^{-}1$.

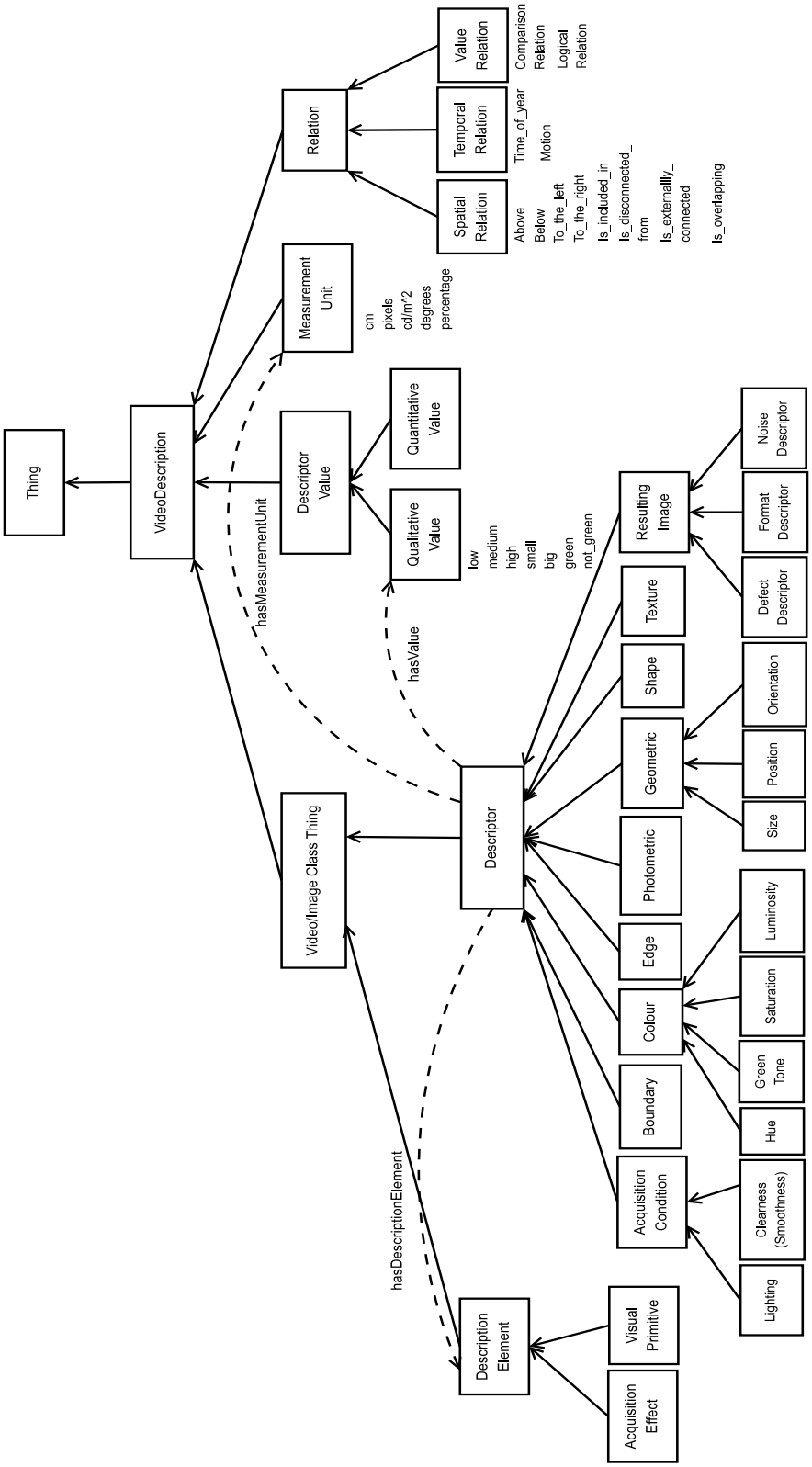


Figure 4.5: Main concepts and relations of the Video Description Ontology.

These concepts have been reused from the Hermes image class model, with the addition that video objects have also been considered. Where appropriate, the image class model was simplified. For instance, the image class is viewed to have three levels of definition in the Hermes ontology: physical, semantic and perceptive. As this does not contribute to providing any meaning for the representation and reasoning within the scope of this work, it was removed from the ontology. Also, the `Relation` class which is a subclass of `Image Class` in Hermes is now moved to be a separate class, as a relationship is not considered as a video or image class but rather a separate entity. The class relations 'hasDescriptionElement', 'hasMeasurementUnit' and 'hasValue' are also additions to this ontology.

Similar to the goal ontology, it contains the binary relation 'is_related_to' to tie video descriptions to relevant goals. For the task that involves detection, the video descriptions that are relevant for this goal is encoded in the 'class_rel' predicate:

```
class_rel(is_related_to, luminosity, detection).
class_rel(is_related_to, clearness, detection).
class_rel(is_related_to, green_tone, detection).
```

These three facts indicate that the descriptors `Luminosity (Brightness), Clearness` and `Green Tone` are related to the goal `Detection`. This ontology is primarily used with the goal ontology in the goal formulation process.

4.6 Capability Ontology

The capability ontology (Fig. 4.6) contains the classes of video and image processing functions and tools. Each function (or capability) is associated with one or more tools. A tool is a software component that can perform a video or image processing task independently, or a technique within an integrated vision library that may be invoked with given parameters. This ontology will be used directly by the planner in order to identify the tools that will be used to solve the problem. As this ontology was constructed from scratch, the METHONTOLOGY methodology [42] was adopted. It is a comprehensive methodology for building ontologies either from scratch, reusing other ontologies as they are, or by a process of re-engineering them. The framework enables the construction of ontologies at the knowledge level, *i.e.* the conceptual level, as opposed to the implementation level. First a glossary of terms, natural language

definitions for the relevant concepts, their synonyms and acronyms were determined. Then a concept taxonomy was built, followed by an ad-hoc binary relations diagram. Next a concept dictionary is built where instances, attributes and relations were determined. Finally these relations, attributes and constants are refined, followed by the definition of rules and axioms.

The main concepts intended for this ontology have been identified as the **tools**, the **functions** that they can achieve and the **performance measures** tied to a combination of known domain descriptions (video descriptions and constraints) for a subset of the tools. The initial set of tools comprised a set of primitive functions derived from the compilation of several vision libraries such as OpenCV [53], Khoros [57] and Pandore [23]. It was decided that the VIP tools should follow some straightforward categorisation while the functions that they can perform should be contained in a more descriptive hierarchy to promote understanding. Each tool is then tied to one or more functions using the binary relation 'canPerform'. The performance measure tied to a tool with respect to a combination of domain descriptions (video descriptions and constraints) were obtained during domain modelling and encoded as rules after the concepts and their relations have been determined for this ontology. The result from this exercise, verified by image processing experts, is the ontology contained in Fig. 4.6. The main concepts include `VIP Tool`, `VIP Operation`, and `Performance Criteria`. For readability purposes, these three main branches of the ontology are provided again (formally) in Appendix G.

The concepts in blue text denote the instances of tools that have been identified after the specification of suitable workflow-composable image processing tools (details of these tools are provided in Chapter 6). It can be seen that there are seven different tools that can perform the function 'Create Background Model'. However, each one of them will perform with a different degree of effectiveness for the same domain description. This is where the performance indication comes into play. For a tool, where known, a description that best fits the domain conditions most suitable for that tool is encoded in a string in the 'hasPerformanceIndicator' predicate. For example, the recommended description for the Gaussian background model is encoded as:

hasPerformanceIndicator(create_gaussian_bg_model, 'Clear and Fast Background Movement').

Then, the list of domain conditions that satisfy this description is contained in the

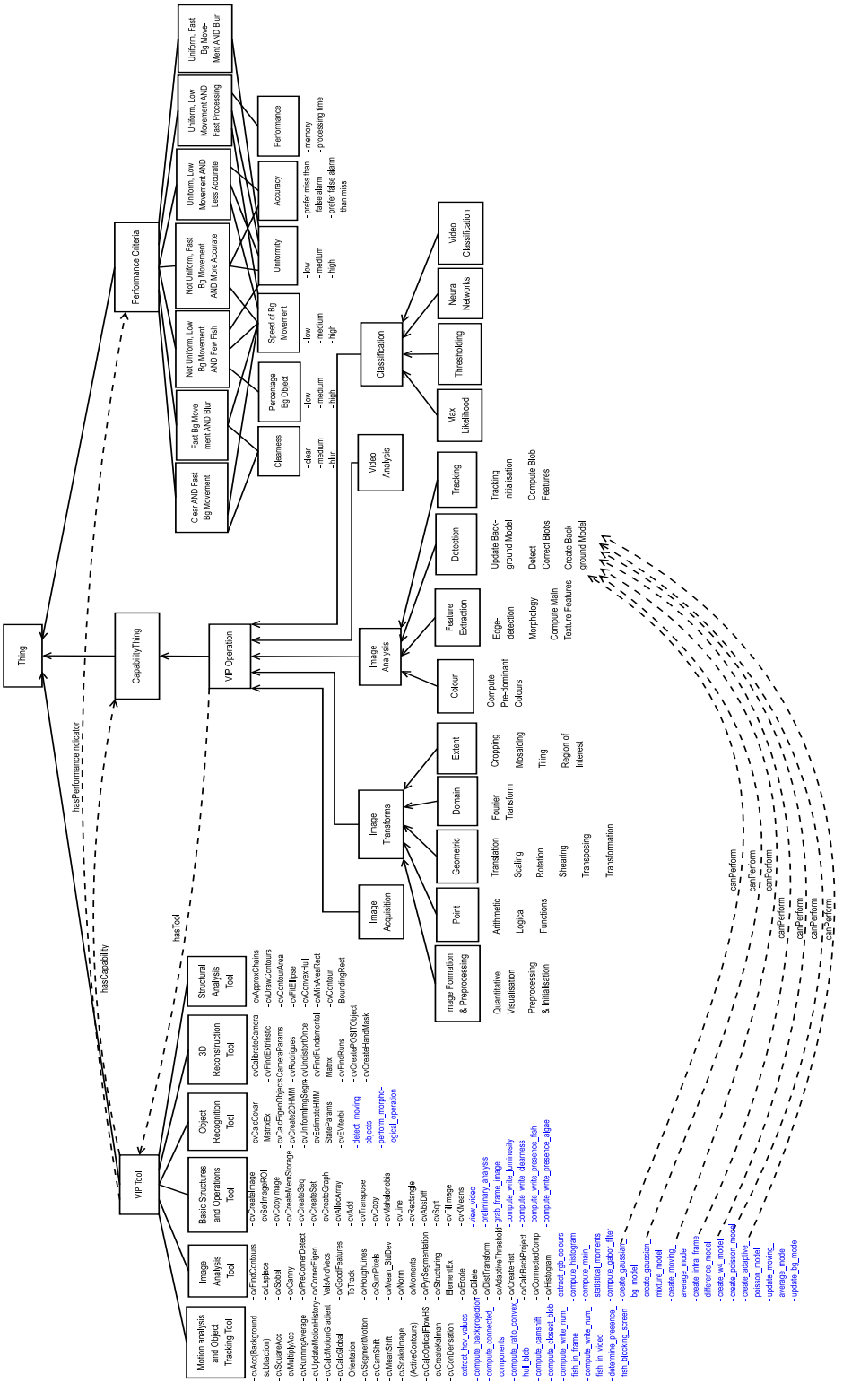


Figure 4.6: Main concepts, relations and rules of the Capability Ontology

predicate ‘instance_att_list’:

instance_att_list(‘Clear and Fast Background Movement’, [clearness(high), bg_movement (high)]).

Based on this, the best tool to perform a task with a set of domain description can be determined. This ontology is used to derive the tool(s) that can perform a primitive task that is identified by the planner. It is also used to provide descriptions and recommendations to the user when more than one tool is encountered to perform a task. Section 4.8 walks through the reasoning mechanism of this process. Chapters 6 and 7 also contain examples of this recommendation and derivation process.

4.7 Example Ontology Use

In this section a few examples of the VIP ontology’s usage is demonstrated. As mentioned in previous sections, it can be used for representation as well as inference. The major roles that have been identified include guiding user and workflow for requirement retrieval (Section 4.7.1), assisting with decision-making for image processing-naive users (Section 4.7.2) and performing consistency checks (Section 4.7.3). These are described in the next three subsections. Finally an example walkthrough of ontology use for fish detection task in a video is described in Section 4.8.

4.7.1 Guide User and Workflow for Requirement Retrieval

During goal formulation, the user is prompted for the task (goal) that they wish to perform. This can be done in two ways. The most conventional way is by presenting all the goals available to the user. However this could lead to information overload if the number of goals available is high (*e.g.* more than five). Hence a more guided way is adopted in this scenario. The main goals (*e.g.* classification, detection, segmentation) are presented to the user to start with. When the user selects one of these, all the subclasses or instances of this goal will be presented for them to make a further selection. This process is repeated until the instance of the desired goal is found. Fig. 4.7 illustrates this abstraction down the goal ontology’s hierarchy.

For example, if the user selects `Classification`, then the next set of tasks presented to them would be `Classify Object` and `Classify Image/Video`. These can be retrieved using the subclass relation in the goal ontology. This process of retrieving

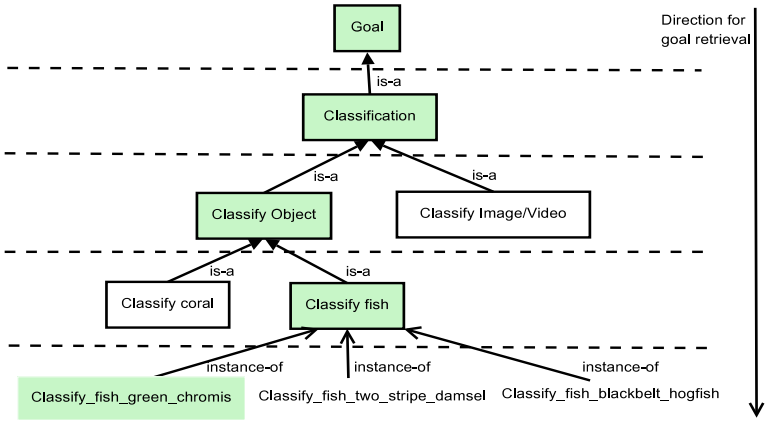


Figure 4.7: Abstraction down the goal ontology to retrieve the goal 'Classify Fish Green Chromis'.

the subclasses of the user-selected goal is repeated until instances are encountered. Thus, when the most specific individual of the goal is encountered and selected, the final user goal is determined. Suppose the user wants all the green chromis fish to be classified, s/he first selects Classification, followed by Classify Object, and finally *Classify_fish_green_chromis*. The reasoning for this goal retrieval is achieved using a simple mechanism consisting of a loop and the relations 'sublass_of' and 'instance_of'. It is summarised in the following pseudo code:

```
Retrieve_goal(G)
{
if G is empty, return []
else
 Goal_list = List of all subclasses and instances of G
 User selects an element g, within Goal_list
 if g is a goal instance
   Return g
 else
   Retrieve_goal(g)
}
```

Once the specific goal is determined, the relevant constraints for the goal could be retrieved. These are encoded in the class relation 'is_related_to'. Thus, only the constraints relevant for the goal are retrieved and prompted from the user. For example the constraint criteria `Accuracy` and `Occurrence` are only relevant for detection tasks, as they describe the level of detection accuracy and the constriction of the number of objects detected. Hence these constraints are not applicable to other tasks. With the help of the goal ontology, the derivation of the goal and constraints are conducted in a systematic and consistent manner, by avoiding unnecessary information overload.

The retrieval of the video descriptions (*e.g.* brightness, clearness and green tone) could also be performed using the same principle. The workflow is then able to formulate the goal and domain description for the user requirement. Section 4.8 provides a concrete example where this mechanism is applied.

4.7.2 Assist Image Processing-naive Users in Decision-Making

Once the goal, constraints and video description have been formulated, the planner generates the solution steps. The capability ontology is used to derive the tool for a particular task using the 'canPerform' relation that relates a tool to the function (primitive task) that it can perform. Sometimes more than one tool can perform a primitive task. In this situation, there will be two courses of actions depending on the planning mode. As will be detailed in Chapter 6, there are two modes of planning; full automatic where no user intervention is required and semi-automatic where user will select the preferred tool from a list of available tools to perform a primitive task. In the semi-automatic case, the user is presented with all the tool options and prompted to select one of them. Having no image processing expertise this would not be an easy decision to make. In this case, the description of the primitive task is obtained from the capability ontology and displayed to the user. Next the tools that can perform the task, along with their recommended descriptions, also derived from the capability ontology, are presented to the user. The recommended descriptions describe the best domain conditions for the application of the tool, verified by image processing experts (see Section 4.6 for an example). In the automatic mode, the system would compute a preferred tool from all the available tools based on the most suitable domain description that match the task. In the semi-automatic mode, the user will be presented with the tool recommended by the system as well. Hence, using the recommended descriptions of the available tools and the tool suggested by the system, the user can make a more informed decision. Experiments that evaluate the level of confidence and learnability in users when selecting tools are described in Chapter 7. Fig. 4.8 shows how the system interacts this information to the user.

4.7.3 Consistency Checking

The VIP ontology is used for various consistency checkings throughout the planning and execution stages of the workflow engine. The relations 'instance_of', 'specialisation_of' and 'subclass_of' allow for simple consistencies such as taxonomical rele-

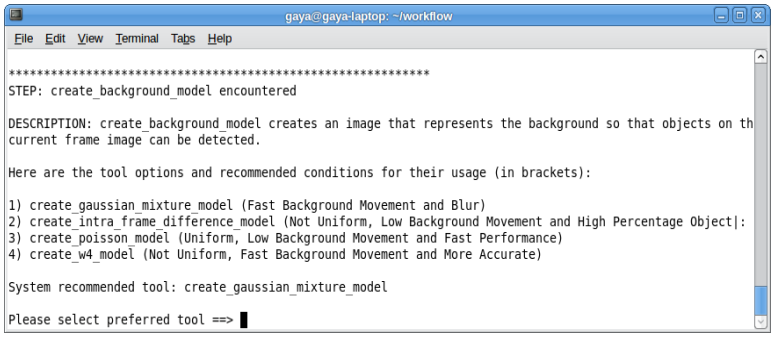


Figure 4.8: Screenshot of system's recommendations to assist user make a more informed decision.

vance between concepts and instances. The axioms defined in the ontology are another source for consistency checks. The 'instance' and 'descendant_of' axioms check that an instance is related to class in a hierarchical fashion. The ontology could also be used to check that a colour-related task is not applicable to a greyscale image, thus avoiding further conflicts. Likewise, an edge detection tool cannot be used to perform colour-related functions.

4.8 Walkthrough

Based on the devised sub-ontologies, a walkthrough on how they are used to provide different levels of vocabulary for the users, vision tools and processes in a seamless and related manner is outlined here. The flowchart in Fig. 4.9 explains how the workflow enactment process takes place. User intervention and points that require ontological usage are emphasised. Some minor steps are omitted from the diagram due to space limitation such as retrieval of input video and planning mode from the user at the start, some intricate details of the planning and the post-planning processes. These are not relevant for this particular section. The walkthrough is explained using the flowchart diagram for an example VIP task.

The user may have a high level goal or task such as *"Detect all the fish in video 1.mpeg"* in mind. One way this could be represented and selected are via the following criterion-value pairs in natural language:

$$[\text{Goal: goal} = \text{detect\_fish}]$$
$$[\text{Constraints: Performance} = \text{memory, Quality} = \text{reliability,}$$
$$\text{Accuracy} = \text{prefer\_miss\_than\_false\_alarm, Occurrence} = \text{all}]$$

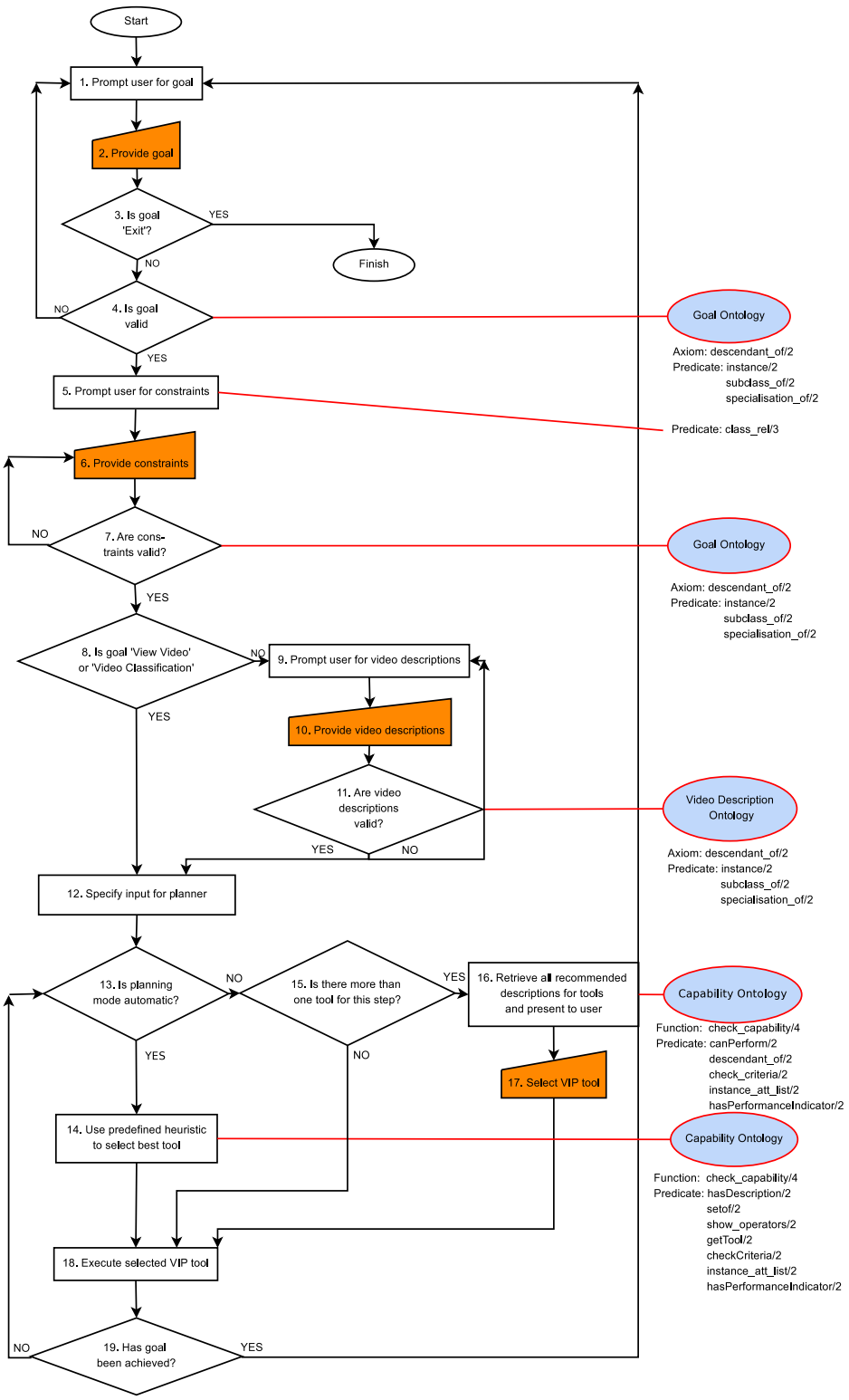


Figure 4.9: Flowchart of the workflow enactor's interaction with the user annotated with ontology use. The red lines indicate points in which ontologies are consulted. Processes in orange denote user-provided input.

[Video Descriptions: Brightness = not_known, Clearness = not_known,
Green Tone = not_known]

The system interprets these values via the use of predicates. The mechanism for the input retrieval is explained as follows. As a first step, the user is prompted for input values for the goal (see Section 6.2.2.1 for an explanation on this). This corresponds to steps 1 and 2 in the flowchart. Once the goal is retrieved, it is checked against the goal ontology (step 4). As explained in Section 4.7.3, the axioms 'instance' and 'descendant_of' are used to check if the goal is indeed one that is valid for the application in question. In this case the instance 'detect_fish' is checked against the class 'goal' to determine if they belong to the same hierarchy.

Next, the related constraints for the goal are determined (steps 5 and 6). These are additional parameters to specify rules or restrictions that apply to the goal. As explained in Section 4.4, the predicate 'class_rel' is used to retrieve the relevant constraint criteria for a goal. For the goal 'detect_fish', the relevant constraints are `Performance Criteria`, `Quality Criteria`, `Accuracy Level` and `Occurrence`, contained in the goal ontology. The user may choose to provide all, some or none of these values. Adopting the same principle used for goal retrieval, constraint values are checked if they are valid (step 7), *i.e.* if they are descendants of the class 'constraint_qualifier'.

Then, depending on the goal, the user is prompted for the video descriptions (steps 9 and 10). For the task 'detect_fish', the descriptions required are brightness, clearness and green tone levels. In essence, for all tasks apart from video classification and video display (that do not require any such information), these criteria will be taken into account from the user. As before, the user may choose to provide all, some or none of these values. The values obtained are checked against the video description ontology (step 11). The checking of validity is again the same as with the goal and constraints, except that the values (instances) are checked against the class 'descriptor' in the video description ontology. In the absence of user information, constraints are set to default values (see Section 6.2.2.1), while video descriptions are obtained via preliminary analysis later on (see Section 5.5.1).

Once the goal, constraints and video descriptions are determined, the formulation of the user's problem is complete and this information will be fed to the planner (step 12). The inner workings of the planner will be described in Section 6.3. Basically, the planner seeks to find steps in the form of VIP tools composed in sequential, iterative and conditional fashions in order to solve the task. At each step of the way, the planner attempts to find a suitable tool by consulting the capability ontology (step 14/16). This

is done using the‘check_capability’ function, described in Algorithm 1 below.

```
check_capability(planning-step S, planning-mode Auto)
If Auto is true
       Retrieve a tool, T that can perform S from capability ontology:
       getTool(S)
       Check that domain-criteria tied to this tool (if any) hold:
       check_criteria(T)
return T
Else
       Display description of S, D to user:
       hasDescription(S, D)
       Retrieve ALL tools that can perform S from capability ontology:
       For each tool ti in Ts
          Retrieve recommended domain descriptions, RD for ti:
          hasPerformanceIndicator(ti, RD)
          Display ti and RD to user
       End for
          Display also system’s recommended tool, ts
          getTool(S)
       Prompt user to select a tool T
Return T

getTool(S)
       canPerform(T, S)
       instance(T, T1)
       descendant_of(T1, tool)
Return T

check_criteria(T)
       If a set of domain-criteria, DC exist for this tool
       hasPerformanceIndicator(T, DC)
          Retrieve the list of preconditions, P for DC:
          instance_att_list(DC, P)
             If all preconditions in P hold
                return DC
             Else return Fail
       Else return ‘no_criteria’
```

Algorithm 1: VIP tool (planning step) selection via the ‘check_capability’ function.

First it retrieves a tool that can perform the planning step. Subsequently, it checks if the selected tool is linked to a list of domain conditions that are deemed suitable for

it according to image processing experts' heuristics. Not all tools are tied to domain conditions. The domain conditions that are suitable for this tool are checked against the current domain conditions. If all of them hold, then the tool is selected for execution. Otherwise it will try to find another tool where such conditions hold, failing otherwise. This is applied when the planning mode is automatic (steps 13 and 14). In semi-automatic mode, user will make this tool selection whenever more than one tool is present to perform a planning step. If there is only one tool that can perform the planning step, it will be selected. When more than one tool is available to perform a planning step, all the tools and their recommended descriptions are displayed to the user who will select one of them (steps 15 and 16). The descriptions are expressed in natural language to ease readability for the user. When a tool is selected, it is applied directly to the video or image in question (step 18). This planning interleaved with execution process continues until the task is solved, *i.e.* when the goal list is empty.

4.9 Conclusion

The formulation of the video analysis problem description and solution could be tackled by modularising them using separate but inter-related ontologies. The three sub-ontologies presented (goal, video description and capability) are comprehensive and describe the important aspects of the video and image processing domain clearly and succinctly. Functions and axioms have provided a way to make inferences that would check for consistencies. They are also easily extensible to add new concepts, instances and relations. This would prove to be useful as it facilitates a means to formalise the video analysis process and promotes reusability of applications. Task modelling was not included in the ontology as it is dealt within the planner which will be described in Chapter 6. The next chapter will present the video and image processing tools that were developed for the purposes of this thesis.

Chapter 5

Video & Image Processing Components and their Representation

The video and image processing (VIP) components constitute the tasks of the workflows composed for the purposes of this research. They are contained in the processing layer of the workflow composition and execution framework outlined in Chapter 3. The components are represented in the process library as primitive processes and the capability ontology as video and image processing tools. They are identified and invoked by the planner that composes and executes the workflows. This chapter begins by presenting the motivations for the construction of VIP solutions using multiple image processing components as opposed to traditional monolithic approaches undertaken by image processing experts. Subsequently the chapter will focus on the methodology for the derivation of a suitable level of granularity for the components using a combination of top-down and bottom-up approaches. Then, their representation and descriptions using an example VIP task involving video classification, fish detection, fish counting and fish tracking will be described. 30 tools were identified and implemented with close collaboration with image processing experts using OpenCV. These were then used to maximal effect in the workflow context for video classification, fish detection, counting and tracking tasks. Only a small fraction of these components ($\sim$10%) are applicable to specific video processing tasks, indicating a high level of reusability for the image processing components for a spectrum of VIP tasks.

5.1 Motivation

Despite being a relatively young field, computer vision has advanced rapidly over the past few decades especially in the branch of image processing[1]. Generally, image processing includes tasks such as recognition, motion analysis, scene reconstruction and image restoration. Recognition tasks typically involves the detection of pre-specified objects of interest while motion analysis includes tasks such as tracking that involves following the movement of a set of points of interest or objects in the image sequence. Both these tasks are particularly relevant for videos. Scene reconstruction aims at computing a 3D model from a set of images in sequence, while image restoration filters out noise such as sensor noise and motion blur from images. How these tasks are conducted within an image processing system will be described next.

5.1.1 Single Executable Approach

Most image processing systems are built depending heavily on the application that they are trying to solve. Broadly speaking, most computer vision systems contain functions that perform the following operations: image acquisition, pre-processing, feature extraction, detection, segmentation and high level processing. The image acquisition process involves transforming real world data into a digital image. This is achieved via the use of image sensors, cameras and other devices. The resulting image is a 2D image, a 3D volume or a video (image sequence). The pre-processing step involves preparing the image data before specific image processing algorithms can be applied to it. This typically involves cleansing the image data by performing noise reduction and contrast enhancement to improve the quality of the image. Features that can be extracted from the image include lines, edges, shape and texture features (such as statistical moments and histogram values).

Detection and segmentation involve finding regions of interest (ROIs) that will be used for further processing. Segmentation often involves the division between different types of regions identified in an image. Many algorithms have been developed to provide efficient detection and segmentation based on the type and quality of the image. Detection often involves the subtraction of an image containing objects with an image without such objects (background model) to obtain the ROIs. A background model is a representation of a static scene without any moving objects in it. This can be done using adaptive or non adaptive methods. Adaptive background models are created just

[1] For clarity, the term “image processing” corresponds to “video and image processing”.

once at the start of the video processing as they are updated online. Two adaptive background model algorithms are the Adaptive Gaussian Mixture Model [120] and Adaptive Poisson Model [98]. Conversely, non adaptive background models will need to be updated (re-created) at each frame of the video. Some non adaptive background model algorithms include the Moving Average Model [53], Gaussian Model [89], W4 Model [45], Intra Frame Difference Model [8] and Poisson Model [98].

Traditionally, the specific implementation of a computer vision system is in the form of a program written in a low-level language, such as C++. Over time, image processing developers realised the value of consolidating the image processing modules within integrated libraries. Due to this, some well-known libraries were developed and made available, such as OpenGL [44], VXL [102] and OpenCV [53]. Many high level languages that were not primarily built with image processing utilities now support image processing capabilities within them, including MATLAB [66] and Java, the former has been used widely among the image processing community. Three such libraries were investigated for the consideration of this thesis. The choices were based on their comprehensiveness and accessibility for the purposes of this research.

5.1.2 Image Processing Libraries

As stated in Chapter 2, research efforts within the image processing community have concentrated on producing sophisticated algorithms for highly specialised tasks. Several multi-purpose libraries that contain low level functions for general use by image processing programmers have been developed and made available. A notable system is Intel's Open Computer Vision library (OpenCV) [53]. OpenCV is an extensive C/C++ based library of programming functions mainly aimed at real time computer vision applications. It contains a wide range of features such as image manipulations, matrix, vector and linear algebra routines, image processing functions such as filtering, edge and corner detection, morphological operations and histograms, structural analysis including connected components, contour processing and several transform functions, motion analysis including tracking, object recognition and also GUI functions to display images and videos. Two key advantages of OpenCV are comprehensiveness and speed of execution, which are crucial for effective and efficient processing. It is also freely available for download and compatible with various platforms.

MATLAB [66] is a high-level language and interactive environment that enables computationally intensive tasks such as matrix manipulation to be performed effi-

ciently. The MATLAB Image Processing Toolbox provides a set of reference-standard algorithms and graphical tools for image processing, analysis, visualisation, and algorithm development. It includes techniques for image enhancement (linear and nonlinear filtering, filter design, deblurring, and automatic contrast enhancement), image analysis, colour image processing techniques, spatial transformations and image transforms. However, one major drawback of MATLAB is its speed of processing that is relatively slower than that of OpenCV, which could hinder efficiency.

Pandore [23] is an open source C++ based library developed at the GREYC Laboratory, France. Any operator from this library is a program performing an operation that cannot be further decomposed. Thus, any complex operation has to be decomposed into a sequence of several operators. Each operator takes, as inputs, images and numerical parameters and produces, as outputs, images and other (non-image) results. It can perform a specific operation on various image formats (pixel image, label image, region map, *etc.*) and inputs and outputs are normalised.

For reasons of speed, comprehensiveness and accessibility, OpenCV was selected over MATLAB as the library for developing the image processing tools. It also has the flexibility of building components from scratch as the operators within it are lower level than those in Pandore. It also does not require the image and video file types to be converted to a custom format as imposed by Pandore.

5.2 Motion Detection System using a Single Executable

As explained in Section 5.1.1, image processing experts solve VIP tasks computationally by using a single executable system, *i.e.* by writing a single program which is compiled into an executable and applied to images or videos. In this section, an example single executable system for motion detection and analysis is described. This will provide an understanding of the processes involved in single executable systems before multiple executable systems are introduced. This single executable system performs video classification, object (fish) detection, counting and tracking tasks for undersea videos [99]. The OpenCV program written for this single executable system was the basis for the derivation of the image processing components to be used in a multiple-executable workflow context, which was implemented for this thesis.

In general, motion detection systems distinguish three levels of processing; pixel, frame and tracking [31]. To illustrate a concrete example, the pixel level processing, frame level and tracking level algorithms for fish detection, counting and tracking tasks

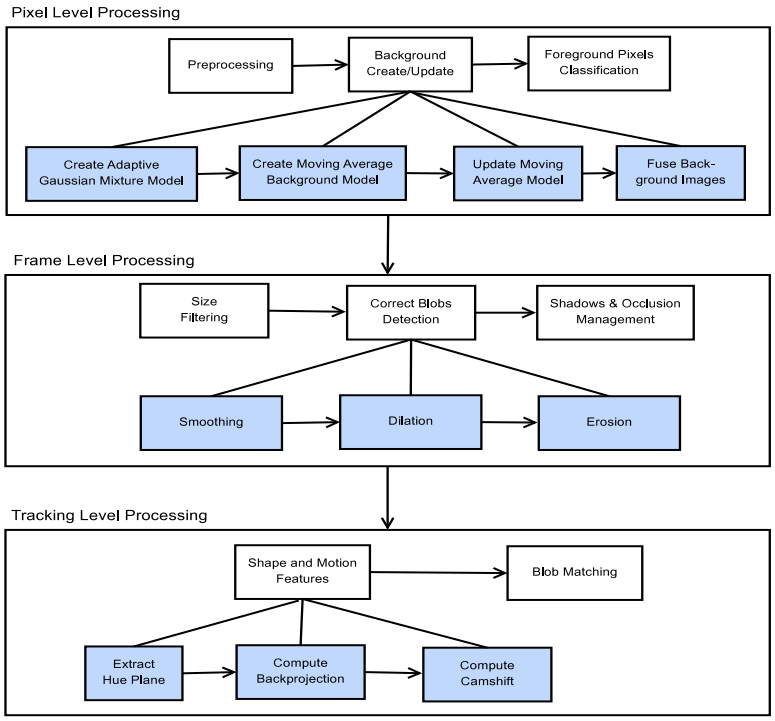


Figure 5.1: Pixel, frame and tracking level processing algorithms for fish detection, counting and tracking task.

are shown in Fig. 5.1. The shaded boxes indicate the specific algorithms that have been identified to perform subtasks at each processing level.

As can be seen from Fig. 5.1, the algorithms at the pixel level are first applied, followed by the ones at the frame level and finally the algorithms at the tracking level. The pixel processing level aims at identifying pixels belonging to objects, hence at classifying pixels as foreground (objects) or background by a comparison with a background model, which is also created at this level. This motion detection system uses a fusion of two background models, Adaptive Gaussian Mixture and Moving Average models. The fusion of the two background models is achieved by finding the intersection between the two background images. Once a background model is created, the foreground objects in the current frame image are determined. This is achieved by removing occlusion and negligible (small) objects. The pixels identified can be visualised as a binary image (with black and white pixels) with the objects represented as white pixels and the background as black pixels. Fig. 5.2 illustrates an example.

The frame processing level aims at analysing foreground pixels for grouping them into defined blobs, excluding the groups of pixels smaller than a certain size (size filtering). Moreover, the objects to be detected should be separated from their shadows

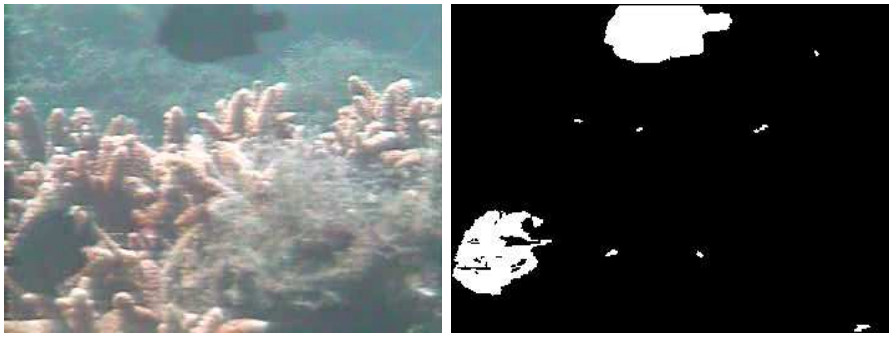


Figure 5.2: Pixel level processing to identify foreground objects (right) from original frame image (left).

and occlusion suppression should be done to separate blobs that represent more than one object. Shape filtering could also be applied to exclude objects of non interest. Basically this processing involves detecting the correct objects (blobs) among all the objects identified from the pixel level processing.

Following the example from Fig. 5.2, once the foreground objects are determined, they will need to be reanalysed to identify the objects that are of interest for the detection task, *i.e.* fish. This is often done via a shape and/or size filtering mechanism. In this example, morphological operations that include a smoothing, followed by a dilation and then an erosion are applied to the binary image produced by the pixel level processing. A shape filtering is also applied where the shape of a fish object is determined via the computation of the area of the convex hull of a blob over the area of the blob itself. Fig. 5.3 illustrates an example frame level processing to detect fish.

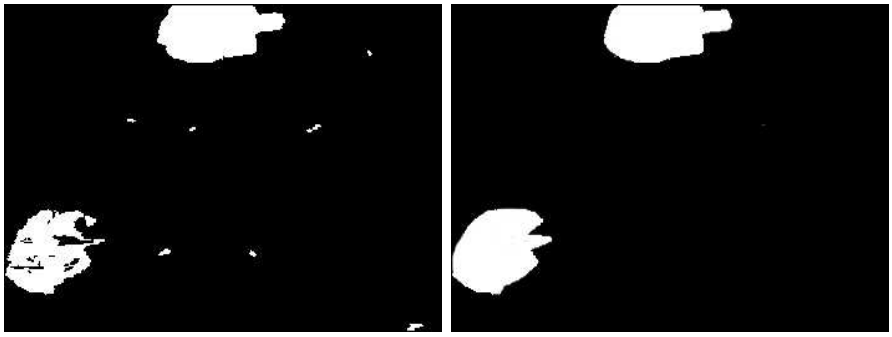


Figure 5.3: Frame level processing to identify blobs (right) from binary image (left) with detected objects.

Finally the tracking processing level aims at achieving, after an appropriate extraction of the blob's features, blob matching to track the objects over time. This involves comparisons between the blob in question with all the blobs in a fixed number of pre-

ceding frames to find the blob that matches it best.

Taking the example from Figures 5.2 and 5.3, once the correct blobs have been identified, the processing is passed to the tracking level. Here, objects in consecutive frames are examined to identify which ones represent the same fish object (tracking). First, the backprojection image of the hue plane is computed. This image is used to predict what a blob will look like in the next frame using the Continuously Adaptive Mean Shift (Camshift) algorithm [15] by returning its centre, orientation and size. Using these three information, the Euclidian distance between the blobs in two frames are calculated to determine the closest matching pairs. This is repeated to compare blobs in a segment of ten consecutive frames. In this way, pairs of blobs that "match" in the segment refer to the same fish object. Fig. 5.4 shows the result of applying tracking level algorithms for fish counting and tracking.

Figure 5.4: Example tracking level processing for fish counting in a video. Sample results for fish detection (left) and fish counting (right). The top number indicates the number of fish in the current frame and the bottom number indicates the total number of fish in the video so far.

Using this single executable motion detection system, a combined top-down and bottom-up methodology was applied to derive a multiple-executable system for video classification, fish detection, counting and tracking. This multiple-executable system is intended to provide the basis for a modular and reusable way to solve VIP tasks.

5.3 Methodology

One of the most challenging aspects of conducting this research was identifying a suitable set of image processing tools that would represent a group of operators in the capability ontology and process library. Typically, an image processing task is solved by writing a program that is compiled into an executable which can be run on an input

video. However, having just one executable would only work on one task or a small subset of image processing tasks. In order to construct such programs automatically, executables of a lower level of granularity would be required. In order to do this, a combined top-down and bottom-up approach was adopted, similar to the method advocated by Uschold and King for ontological building approach [110]. First, the program code was inspected thoroughly and tasks were broken down in a top-down manner. This involved breaking down the steps used in solving the task into meaningful blocks or components. Subsequently, however, the bottom level tasks were grouped by procedure to provide a coarser level of granularity that was more manageable. This methodology has been used effectively to accomplish the derivation of image processing components for this thesis. The approaches are outlined in more detail below.

5.3.1 Top-down Approach: Function Calls as Primitive Tasks

Initially, a top-down approach was adopted whereby operators were grouped by primitive processes in the image processing program. The image processing task can be seen as the high level goal that is decomposable into several major subtasks that are in turn decomposable into further subtasks until primitive processes are encountered. As mentioned in Section 5.1.2, OpenCV (version 1.0) was selected as the basis for the image processing code after surveying three computer vision libraries. In a typical OpenCV program, the primitive processes correspond to function calls, assignments, arithmetic and logical operations. This tedious process involved separating variable declarations, headers and function prototypes from the body of the program, and then breaking down the program body into blocks of major subtasks, taking into consideration conditional statements (*e.g.* `if-then`) and loops (*e.g.* `for, while`). Once the major subtasks were identified, they were further decomposed until primitive level.

The hierarchical decomposition was done on a program of approximately 1000 lines of code performing a video classification, fish detection, counting and tracking task [99], which was the most complex task used for the thesis experiments. This method broke the one big task into its primitive level operators. Among the major features or modules that were determined included pre-processing that involved video capture and frame image grabbing, a main loop that processes each frame which involved fish detection, extraction and tracking procedures, and a classification and output phase that computed the final results, and created an output video containing these results. Fig. 5.5 shows some example operators derived from applying the top-down

approach to the OpenCV program. These are contained at the bottom layer of the diagram and each can be achieved using a single OpenCV or C++ function call.

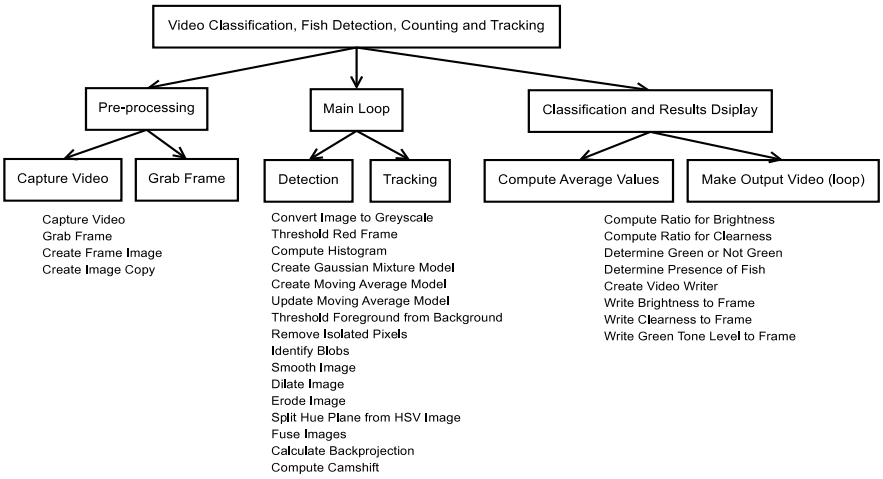


Figure 5.5: Using top-down approach to identify some image processing operators for video classification, fish detection, counting and tracking task.

This exercise yielded 85 unique primitive processes in the process library that were encoded as operators in the capability ontology. When run on a one-minute clip containing 300 frames, 69,011 steps or operator invocations were produced. While the top level goals and their immediate subtasks (shown in white boxes in Fig. 5.5) provided an intuitive representation of the image processing tasks, the bottom level tasks or operators were too fine grained and did not provide a manageable level to work with. They were also too technical for an image processing-naive user to comprehend and make decisions upon (*e.g.* Split Hue Plane from HSV Image). Hence it was decided that some of the low level tasks should be merged to produce a coarser level of granularity, in order to provide a more manageable level for users (and the system) to work with.

5.3.2 Bottom-up Approach: Grouping of Function Calls

Having all the primitive level tasks at hand, they were further packaged where possible to obtain operators with a more suitable level of granularity. This involved grouping the bottom level processes (primitive tasks) by procedure. For the most part, the primitive tasks were grouped to represent the subtask one level immediately above them (see Fig. 5.6). This exercise yielded 30 operators, termed as *independent executables*, that were much more manageable to work with. Description for each independent executable is provided in Section 5.5, their technical description can be found in Appendix A.

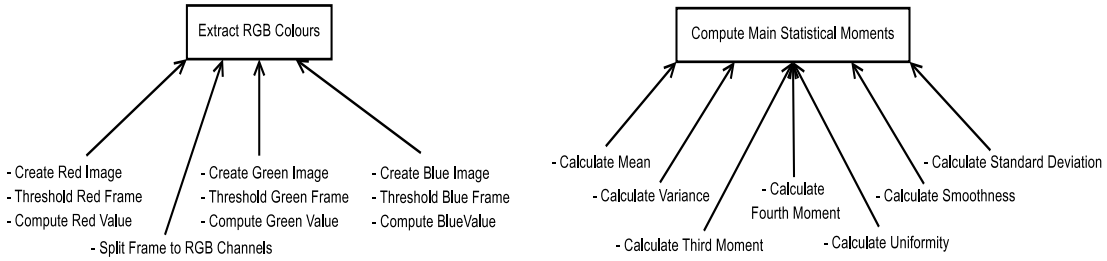


Figure 5.6: Application of bottom-up refinement to derive the executables 'Extract RGB Colours' and 'Compute Main Statistical Moments'.

The advantage of this bottom-up refinement approach has led to the identification of modules that could be reused for most video processing tasks. In addition, the executables provided a more intuitive representation of the video/image processing tasks than their primitive level counterparts. For instance, in Fig. 5.6, the independent executable Compute Main Statistical Moments which was derived by merging primitive tasks Calculate Mean, Calculate Variance, Calculate Third Moment, Calculate Fourth Moment, Calculate Uniformity Calculate Smoothness and Calculate Standard Deviation, is a more compact and concise concept to represent a subtask to compute the mean, standard deviation and other statistical moments of an image.

Care was taken so as not to merge some tasks that need to be invoked independently into higher level subtasks. For example, in order to perform the classification of the video, the executables were developed independently as Compute and Write Average Luminosity, Compute and Write Presence of Fish and Compute and Write Presence of Algae were developed independently. This then does not impose the classification task to include all of these criteria to be classified. Compute and Write Presence of Fish, for instance is not required when performing only video classification task, while it is required when performing video classification combined with fish detection, counting and tracking tasks. Hence the procedure involved thorough and repeated discussions with image processing experts in order to produce the most suitable set of operators.

A further refinement to reduce the number of steps included incorporation of loops within the operators where the number of iterations in the loop were known already or could be determined at run-time. With this reduction of almost threefold in the number of operators from 85 to 30, a sample run on the same one-minute clip of 300 frames tested on the operators from the top-down approach now yielded 8706 execution steps, a reduction of almost eightfold in the number of steps [76].

5.4 Implementation Refinements

The workflow components, in particular the process library, underwent modifications with the application of the bottom-up refinements after the initial top-down method. In this section the implementation issues pertaining to the components in light of the above modifications are discussed.

5.4.1 Suitable Independent Components

By using the combined top-down and bottom-up approaches outlined above, 30 independent components (see Section 5.5) were developed using OpenCV with careful considerations of the input, output, storing and referencing the output of each executable. It was decided that for each input video, a directory with the video filename is created, called the video directory. Under the video directory a set of children directories for the frames that have been processed are created, called frame directories. Where values are strings or numbers, they are stored in text files in the frame directory. This mechanism not only ensures that the values are not lost, but is also useful for debugging purposes should there be problems with any part of the system, where these values could be inspected for errors. Images and videos are also stored in frame and video directories respectively.

5.4.2 Modification to Process Library

The process library, which initially contained 85 primitive processes is now reduced to 30 independent components. This setting is not just easier to manage, the independent components also represent more meaningful tasks that could be reused for various other image processing tasks. The newly specified executables are defined according to their inputs, outputs, pre-conditions, effects and post-conditions in agreement with image processing experts. In addition to these, the process library now contains 28 methods to represent non primitive tasks that are further decomposable, and includes iterations and conditional statements. These will be described in Chapter 6. The code representation of the process library is contained in Appendix A.

5.4.3 Refinements to Ontologies

The goal, video description and capability ontologies that were created to contain concepts and instances were refined to include the changes to the process library. Most of

the work done prior to the refinements involved the population of the goal, video description and capability ontologies with known instances of tasks, domain description and video processing operations and tools that can carry them out. With the modification above, the capability ontology is now revised to contain the independent executables as operators (or tools) as opposed to the primitive level function calls as operators. It is also enriched with performance level information for a set of the tools where this information is known using image processing experts' heuristics. For instance, the video descriptions and/or constraints that best fit each background model are now incorporated into the capability ontology. The suitable conditions for each type of background model are listed in Section 6.5.2. These are represented in the capability ontology as a conjunction of logical statements and tied to the respective tool with a performance measure. This will then enable a performance-based selection during the planning process which will be described in Chapter 6.

5.5 Video & Image Processing Tools/Executables

This subsection presents the 30 independent executables that have been identified using the methodology outlined in Section 5.3. Repeated discussions with image processing experts resulted in the refinement of the processes and the modification of the ontologies with respect to these changes. Section 5.6 presents several examples to demonstrate the use of the tools. Appendix A provides a table with the inputs, outputs and related ontologies, where appropriate, for each tool. As stated in section 5.4.1, a directory is created for each video from the present working directory, known as the *video directory* during execution, *e.g.* for video 1.mpeg, directory Video_1.mpeg/ is created. In the video directory, a separate directory is created for each frame processed within that video, which will be referred to as the *frame directory*, *e.g.* the frame directory for the second frame of video 1.mpeg is Video_1.mpeg/2/. A third directory to store the frame images will be used to create the final video containing the result of the video processing task. This is called the *output directory*, *e.g.* for video 1.mpeg, the result frames are stored in Video_1.mpeg/Output/.

5.5.1 Pre-processing and Initialisation

1. View Video

   The aim of this executable is to display a specified video to the screen, determine

its frame rate, number of frames and creation date, and compute candidate background images from it. The directory BackgroundImages/ is created in the video directory and populated with the candidate background images. The basic properties are saved in a text file view_video.dat in the video directory. This module is included as a video processing task in itself for users who wish to view and capture basic properties of a video. Otherwise it is mainly used to display the video containing the result of a video processing task.

2. Preliminary Analysis
   This module is responsible for capturing the initial video description of the video. A small motion detection operation is also performed to identify background movement, *e.g.* plants. Given a video file, it retrieves the initial brightness, clearness, speed of movement, percentage background object, background movement and initial texture features (variance, skewness, uniformity, entropy). These features are saved in a text file, prelim.dat in the video directory for further manipulation and will be verified with the concepts and instances contained in the video description ontology. This component is generic and can be applied to any video.

3. Grab Frame Image
   This component takes a video file and a frame number as input, retrieves the image for that frame and stores it in the video directory for further processing. This executable is essential for almost all video processing tasks because each frame of the video is processed using its image representation. This component is generic and can be applied to any video.

5.5.2 Compute Predominant Colours

Component 4 defines one way of computing predominant colours in an image.

4. Extract RGB Colours
   Given a frame image, this executable extracts the red, green and blue channel images of a frame. The resulting channel images are stored in the frame directory. This component is generic and can be applied to any video. An additional channel that could be derived from this executable is the yellow channel. It is computed using an arithmetical and logical combination of the red, green and blue channels and can be used for computing the video quality.

5.5.3 Compute Main Texture Features

Components 5–7 describe functions that are related to the computation of texture features. Texture gives information about the spatial arrangement of color or intensities in an image or selected region of an image.

5. Compute Histogram
   This component takes in a frame image, computes its histogram value and image representation. The histogram values are saved in a text file, Hist_Array_<frame_number>.dat and the image representation is saved in a binary file (jpeg), Hist_<frame_number>.jpg, both in the frame directory. This component is generic and can be applied to any video.

6. Compute Main Statistical Moments
   Based on the histogram values computed in component 5, this module determines the main statistical moments such as mean, variance, third moment, fourth moment, entropy, uniformity and smoothness for the histogram. The statistical values are stored in a text file, Hist_Array_<frame_number>_stat.dat in the frame directory. This component is generic and can be applied to any video.

7. Compute Gabor Filter [27]
   A Gabor filter is a linear filter used in image processing for edge detection. This component applies a Gabor filter to a complex image made up of a real and an imaginary part. The absolute value of the complex image is computed, followed by the mean and variance, which are texture features. For four angles and three scales, this will yield 12 mean-variance pair values, so for each image 24 values will be extracted. These features could be used for the detection of coral reef, for instance. This component is generic and can be applied to any video.

5.5.4 Perform Detection

Components 8–18 describe functions that are related to detecting objects in a video.

5.5.4.1 Create Background Model

Components 8–14 describe seven different background model types that have been identified. These are broadly divided into adaptive methods (components 9, 10 and 14) and non-adaptive methods (components 8, 11–13). Adaptive methods are applied only

once per video. Non-adaptive methods will need to be reapplied for each frame of the video. All these components are generic and can be applied to any video.

8. Create Gaussian Background Model [89]
   Given a frame image, this module creates a background model represented by 2 images; foreground and background. It also stores the background model in a directory within the video directory. This background model is good for videos where movement of background objects vary slightly and is thus **not** suitable for videos with *waving trees*[2].

9. Create and Apply Adaptive Gaussian Mixture Model [120]
   Similar to component 8 above, the background model created by this executable is represented by two images and stored in a directory within the video directory. In addition, this module overcomes the limitation of component 8 and is suitable for videos with *waving trees*. This component also detects moving objects in the current frame image and stores the resulting binary image in the frame directory.

10. Create Moving Average Model [53]
    This module takes in a frame image, a directory to store the resulting background model and a *learning speed (alpha)* and creates an image that represents the background model. The advantage of this algorithm is that it does not require a learning phase. Alpha is dependent on changes (*e.g.* lighting, speed of movement), which could be obtained from 2 (Preliminary Analysis).

11. Create Intra-Frame Difference Model [8]
    This module only requires a directory where the background model needs to be stored. The background model is represented by two matrices containing the mean of the pixels of the frames stored in the buffer and the maximum variation of two consecutive pixels of the frames stored in the buffer. This algorithm overcomes the limitations of component 8 but is problematic for videos with impulsive noise (*e.g.* salt and pepper noise).

12. Create W4 Background Model [45]
    This module also only requires a directory where the background model is to be stored. The background model is represented by three matrices containing the mean of the pixels of the frames stored in the buffer, the maximum values of the

[2]Movement of non interesting objects such as leaves, trees, algae, *etc.*

pixels of the frames stored in the buffer and the minimum values of the pixels of the frames stored in the buffer. This algorithm overcomes the limitations of components 8 and 11 and is particularly suitable for videos with low changes in luminosity (*e.g.* indoor videos).

13. Create Poisson Model [98]
    As with components 11 and 12 this module takes in a directory where the background model is to be stored. The background model is represented by two matrices containing the mean and standard deviation of the Poisson distribution calculated from the weighted mean of the pixels' histogram (λ). This algorithm is particularly suitable for videos with uniform background colour (*e.g.* blue water, tar road).

14. Create and Apply Adaptive Poisson Model [98]
    Similar to the three components above, only a directory name is required to store the background model. The background model is represented by two images; foreground and background. This algorithm is similar to component 13, additionally it can manage colour variation with light changes. It also detects moving objects in the current frame image and stores the resulting binary image in the frame directory.

5.5.4.2 Update Background Model

Component 15 updates the moving average model. For the non-adaptive background models, the background models are recreated, whilst for the adaptive ones (Adaptive Gaussian Mixture model and Adaptive Poisson model), the background model is only created once in the beginning using candidate background images derived from component 1 (View Video). This component is generic and can be applied to any video.

15. Update Moving Average Model
    This component takes in a frame image and an existing moving average background model (first created using component 10) to create a new background model (1 image). It utilises absolute subtraction between pixels.

16. Fuse Background Images
    This executable fuses two background images using the logical `AND` operator. A scenario where this component is useful is when none of the seven background models is suitable for a given set of domain description and constraints. In this

case, a background model is created using the fusion of the Adaptive Gaussian Mixture Model and the Moving Average Model.

17. Detect Moving Objects
    This module creates an image with identified blobs from a frame image and a non-adaptive background model (components 8, 11–13). The result is a binary (black and white) image. The algorithm works by removing occlusion and small objects. It also utilises statistical or derivative algorithm based on the type of the background model. This component is generic and can be applied to any video.

5.5.4.3 Detect Correct Blobs

Component 18 is responsible for detecting the blobs that represent the objects of interest and eliminating all others.

18. Perform Morphological Operation - Smoothing, Dilation and Erosion
    This operation is only applicable to non adaptive background models. Given an image with identified blobs (*e.g.* from component 17), an image with potential detected blobs (of fish) is returned. First, a median filter with a 17x17 kernel is applied, followed by 15 iterations of dilation and 10 iterations of erosion. This component is applicable to fish detection tasks only.

5.5.5 Perform Tracking

Components 19–26 provide functions to perform tracking on objects that have been identified via perform detection.

5.5.5.1 Tracking Initialisation

Executables 19 and 20 are components of tracking initialisation.

19. Extract HSV values
    This component extracts the images for the hue, saturation and value channels for a given frame image. The images are stored in the frame directory. This component is generic and can be applied to any video.

20. Compute Backprojection
    This module depends on component 19 as it takes in a hue channel to create a histogram of the hue channel, which is then used to create an image which

represents the backprojection of the hue plane. The backprojection image is stored in the frame directory and will be used for tracking purposes. The hue plane is used because it gives the most useful colour information for an image. This component is generic and can be applied to any video.

5.5.5.2 Compute Blob Features

Components 21 and 22 are used to compute blob features.

21. Compute Connected Components and Area Ratios of Convex Hulls over Blobs
    This module takes in an image with potential blobs (*e.g.* from component 18) and returns an image with the detected blobs (of fish) and also the total number of blobs in that image. It also calculates the ratio between the area of the convex hull of a blob and the area of the blob itself. This component is executed over all blobs in an image. This component is applicable to fish detection tasks only.

22. Compute Camshift
    This module predicts what a blob will look like in the next frame. Given the backprojection of a hue plane (*e.g.* from component 20), an image with blobs (*e.g.* from component 21) and the number of blobs in the image, this algorithm draws a bounding box around each blob present and returns the centre, orientation and size of each blob. This information is stored in a text file in the video directory. This component is generic and can be applied to any image.

23. Compute Closest Blob
    This component is responsible for finding the minimum Euclidian distance between the blobs in two frames. Thus, given the centres, orientations and sizes of the blobs in two frames, the corresponding closest blob in the second frame for each blob in the first frame is determined. It is executed in a loop to compare blobs in a segment of consecutive frames (*e.g.* 10 frames). The pairs of closest blobs between the two frames are stored in a text file, closest_blob.dat. This component is generic and can be applied to any video.

24. Compute and Write Number of Fish in Frame and Video
    This module takes in the current frame image, an array of minimum distances (*e.g.* from component 23), the ratio between the blob area and frame area (*e.g.* from component 21) and computes the number of fish in the current frame and adds it to the total number of fish calculated in all the frames up to this frame.

It writes the number of fish (blobs) in the frame and in the video so far onto an output frame and updates the total number of fish in a text file. Both the outputs are stored in the frame directory. This component is generic and can be applied to any image/video.

25. Determine Presence of Blob Blocking Screen
    Given the area of the blob and the area of the frame, this module will return true (1) or false (0) to indicate if a blob is blocking the screen. The threshold value for the ratio between the blob and frame is set to 70% for the blob to be blocking the screen. This component is generic and can be applied to any image/video to determine if a blob is blocking the screen by a factor of 70% or more.

5.5.6 Perform Video Classification

Components 26–30 are related to video classification functions. All these components are generic and can be applied in any video processing context.

26. Compute and Write Average Luminosity
    This module is responsible for determining if the video is “Bright”, “Medium” or “Dark” based on its average luminosity value, calculated using the mean and 3rd moment. The value is written onto the output frame (an image), which is stored in the output directory.

27. Compute and Write Average Clearness
    This module is responsible for determining if the video is “Clear” or “Blur” based on its average clearness value, calculated using the variance, fourth moment, entropy and uniformity. The value is written onto the output frame (an image), which is stored in the output directory.

28. Compute and Write Presence of Fish
    This module is responsible for determining if the video has “Fish” or “No Fish” based on the number of fish in the video. The value is written onto the output frame (an image), which is stored in the output directory.

29. Compute and Write Presence of Algae
    This module is responsible for determining if the video is “Green” or “Not Green” based on its green channel value. The value is written onto the output frame (an image), which is stored in the output directory.

30. Write Frames to Video

    Given a specified directory, an output video is created using all the frames in that directory with .jpg or .jpeg extensions sorted by name, *e.g.* 1.jpg–50.jpg. This is used to create the output video containing texts indicating classification and counting results and boundary boxes around detected objects.

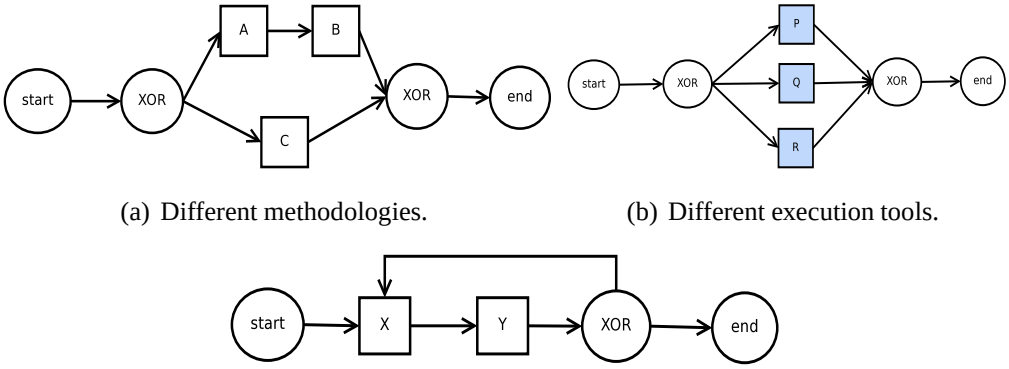
5.6 Selected Process Models

A selection of the process models using the image processing tools specified in the previous section are presented here. They serve to provide an intuitive display of the processes involved in solving VIP tasks (via visual representations) before the planning mechanism is described in the next chapter. The process model diagrams in the rest of this chapter and thesis should be read in accordance with the conventions provided by FBPML in Section 4.3.1. Every process model begins with a `start` node and terminates with an `end` node. Processes can be decomposed to one or more subtasks. A process that can not be decomposed further is a primitive process and will be shaded.

A process model for an image processing task describes the process logic for that task, which can be depicted at a higher level or a lower level of computation. When executed, a process model is instantiated with input values that consist of strings that represent numerical parameters, directories, filenames, images and videos, *etc*. Thus one process model can be instantiated into many execution chains based on different input values, as well as different process logics due to the preconditions that hold for that process logic. One advantage of using process models is that they highlight **decision points** in the flow of processing clearly.

Three main types of decision points have been discovered in this thesis. The first type is a branching point that contains different methodologies for performing the same task. For example, in Fig. 5.7(a), the particular task of this process model can be achieved either by performing A and B or by performing C. The second type is a branching point where different execution steps or tools are selected for a task. This happens at the leaf node of a process model. For example, either leaf (primitive) processes P, Q or R can be selected to perform the task in Fig. 5.7(b). The third type is a natural branching point that the process logic requires, *e.g.* loops. In Fig. 5.7(c), X and Y will repeat until its termination condition is met, in which case it will end.

Parallel processes are contained within the `AND` split-joint construct. These are written sequentially in the program but parallelised here as their executions are indepen-



(a) Different methodologies. (b) Different execution tools.

(c) Natural branching.

Figure 5.7: Three main types of decision points for video processing tasks.

dent of one another. *All* the processes within the `AND-split` construct must be executed. When 2 or more processes branch out of an `OR-split` junction, all the subtasks within it are attempted for execution in a sequence. Failing that the next permutation of subtasks are tried for execution. A typical case involves two subtasks, whereby at first both subtasks are attempted for execution. If both are possible, then both will be selected for execution. Otherwise, the first subtask only is selected for execution. Should that fail, the second subtask only is selected for execution. The next two subsections present the process models for two example VIP tasks used for this thesis.

5.6.1 Video Classification

A video classification task that retrieves domain features such as the video's average brightness, clearness and green tone levels is particularly useful, especially for filtering out videos that are too dark, too bright or blurred from further analysis. In some types of videos, the saturation of certain colours affect the quality of the video, or indicate the presence or absence of certain features. In undersea videos, for instance, high green tone levels indicate that the camera lens will need to be changed due to presence of algae (greenish substance in water). This video classification task could be modelled using the top level plan outlined in Fig. 5.8.

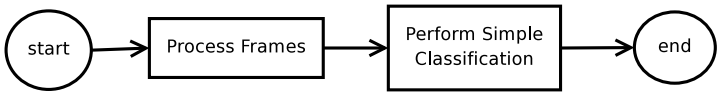


Figure 5.8: Top level plan for 'Perform Video Classification'.

The two main steps Process Frames and Perform Simple Classification are composite processes, they could be further decomposed into lower level subtasks. Fig. 5.9 shows the steps involved in Process Frames.

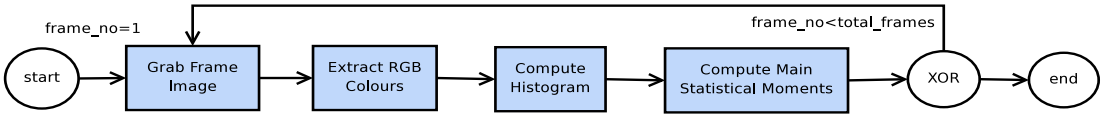


Figure 5.9: Process model for 'Process Frames'.

The XOR construct indicates that this involves looping through the frames to grab the current frame's image, compute the primary (RGB) colours, histogram and main statistical moments accumulatively. These values may be determined using the executables Grab Frame Image (Section 5.5.1), Extract RGB Colours (Section 5.5.2), Compute Histogram and Compute Main Statistical Moments (both in Section 5.5.3). The loop, however, does not need to iterate over all the frames in the video clip. The method Skip Frames (introduced in Section 6.3.5.3) controls this. A condition to determine if all or only a fraction of the frames would depend on the value for the constraint qualifier Performance, which indicates whether the processing time should be minimised (but the result would be less reliable) or the reliability of the result should be prioritised (but takes more time). If the performance qualifier is Processing Time then only a fraction of the frames is used in the loop for faster processing time, otherwise all the frames will be taken into account, providing a more reliable solution but requiring more processing time. Section 6.5 describes this.

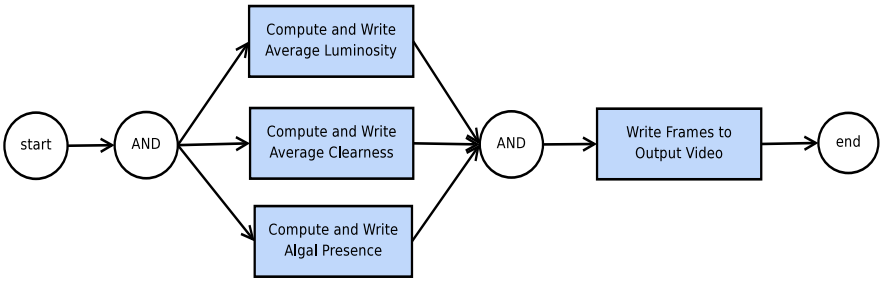


Figure 5.10: Process model for 'Perform Simple Classification'.

Fig. 5.10 shows the steps involved in the subtask Perform Simple Classification. Here, the values accumulated from Process Frames are used to compute the average brightness, clearness and green tone values. The brightness, clearness and green tone classification values are written on to the output frames which are then used to cre-

ate the final output video (using the component Write Frames to Video). Overall, the complete plan for this task could be specified in more detail as contained in Fig. 5.11.

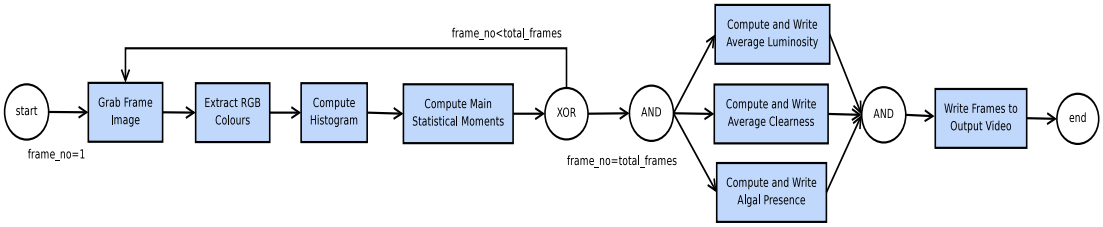


Figure 5.11: Top level plan for 'Perform Video Classification' in detail.

An example plan trace containing the VIP tools and their parameters obtained and executed by the planner for a video classification task is the following:

```
grab_frame_image videos/1.mpeg 1
extract_rgb_colours Video_1.mpeg/1
compute_histogram Video_1.mpeg/1
compute_main_statistical_moments Video_1.mpeg/1/Hist_Array_1.dat
grab_frame_image videos/1.mpeg 2
extract_rgb_colours Video_1.mpeg/2
...
compute_histogram Video_1.mpeg/300
compute_main_statistical_moments Video_1.mpeg/300/Hist_Array_300.dat
compute_write_average_luminosity Video_1.mpeg/
compute_write_average_clearness Video_1.mpeg/
compute_write_presence_algae Video_1.mpeg/
write_frames_to_video Final_1.mpeg.avi Video_1.mpeg/Output/
```

This corresponds to the same set of function calls invoked by the single executable system. For a video with 300 frames, this yields 1204 planning steps or workflow tasks (300 frames x 4 steps and 4 final steps).

5.6.2 Classify Video, Detect, Count and Track Fish

The previous task involved the accumulation of texture features and statistical moments in order to calculate average values for video classification purposes. The example in this section performs the same task, however, it also involves the detection of an object of interest in a video, counting the number of objects in the video as well as tracking it. Firstly the top level plan for this task is given in Fig. 5.12.

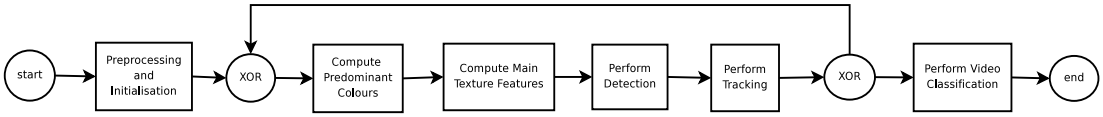


Figure 5.12: Top level plan for 'Perform Video Classification, Fish Detection, Counting and Tracking'.

The whole task is made up of six main subtasks and contains one main loop that executes over all the applicable frames in a video. These subtasks correspond to the sub headings in the previous section; Pre-processing and Initialisation, Compute Pre-dominant Colours, Compute Main Texture Features, Perform Detection, Perform Tracking and Perform Video Classification. Pre-processing and Initialisation consists of View_Video and Preliminary_Analysis. A few of these are looked at in more detail next.

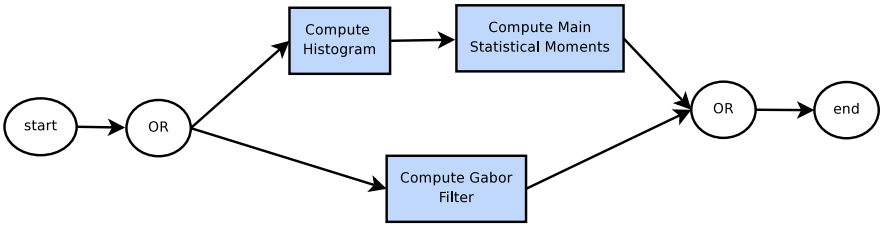


Figure 5.13: Process model for 'Compute Main Texture Features'.

Compute Main Texture Features consists of sub processes that may be executed in parallel (given by the OR construct). The computation of the main statistical moments is only possible after the computation of the histogram. While the computation of the Gabor filter may be performed independently.

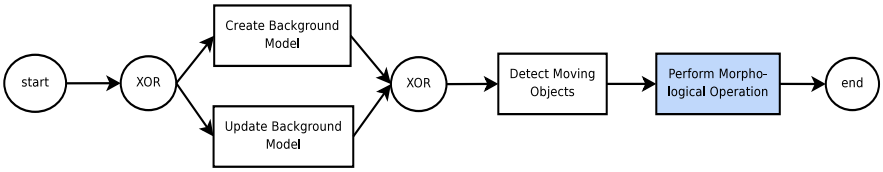


Figure 5.14: Process model for 'Perform Detection'.

Perform Detection is one of the main subtasks of the whole process and will be applied in a loop that takes in one frame of the video at a time starting from the first frame.

There are seven algorithms in which a background model can be created. These are given by the algorithms in Fig. 5.15. Only one of them is selected for a video (illustrated by the XOR notation). The selection of algorithm is dependent on suitable

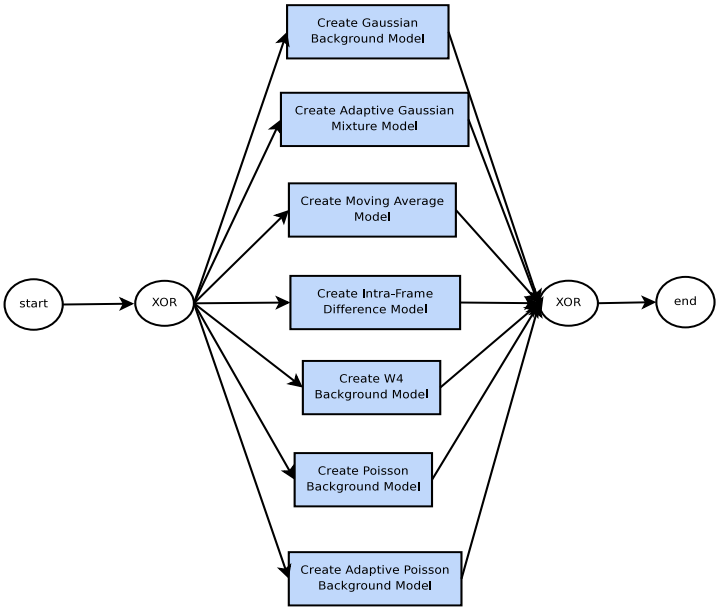


Figure 5.15: Process model for ‘Create Background Model’.

video description and constraints, which will be described in Section 6.5.2. There are

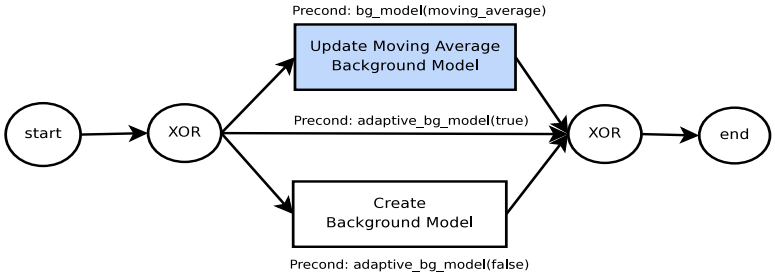


Figure 5.16: Process model for ‘Update Background Model’.

two ways in which a background model may be updated. For the moving average background model, a specific algorithm is required, while for any other non adaptive background model, the background model is recreated. Adaptive background models do not need any updating. Fig. 5.16 illustrates this flow of choices.

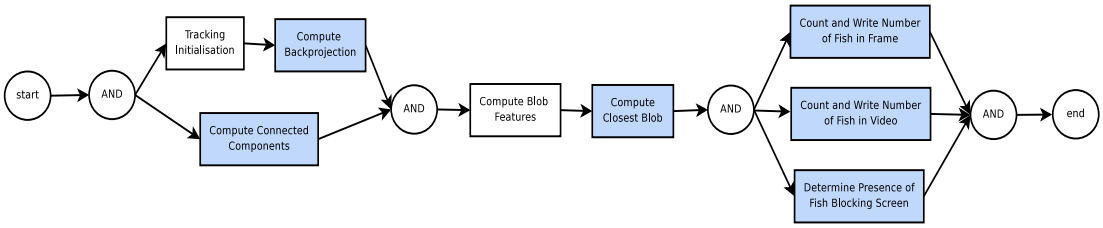


Figure 5.17: Process model for ‘Perform Tracking’.

The process model diagram for Perform Tracking is given by Fig. 5.17. The first block of processing consists of processes that can be run in parallel (indicated by the `AND-split-joint` construct). Tracking Initialisation consists of two primitive steps, Extract HSV Values and Compute Backprojection. Then this is followed by the non primitive process Compute Closest Blob (Fig. 5.18). All other subtasks are independent components and need no further decomposition.

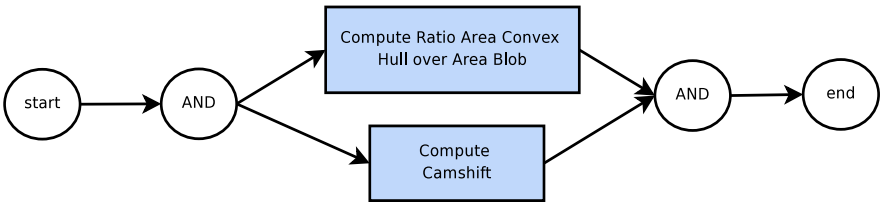


Figure 5.18: Process model for ‘Compute Blob Features’.

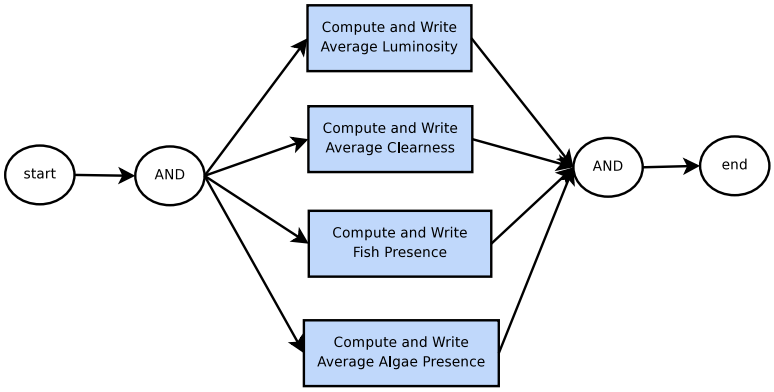


Figure 5.19: Process model for ‘Perform Video Classification’.

Finally the video classification is made up of a set of classification features that need to be determined and written to the output frames. These subtasks are not dependent on one another and could be run in parallel (see Fig. 5.19).

5.7 Concluding Remarks

A set containing 30 video and image processing tools have been developed using OpenCV based on a combination of top-down and bottom-up approaches to be used for the purposes of this thesis. They do not attempt to provide all the possible functions for all the VIP tasks, but a subset of functions that are useful for a typical set of VIP tasks that include video classification, object detection and object tracking tasks. The

range of tasks that they can be used for include basic ones (*e.g.* displaying a video), frequently used functions (*e.g.* computing texture features) and complex ones (*e.g.* video classification, detection and counting). 27 out of 30 of these have been identified as reusable with respect to generic video processing tasks by image processing experts, under the assumptions that the videos are in mpeg format, the images are in jpeg format and the input values they depend on (text files, images and videos) are specified in the process library. The image processing components were developed through close collaboration with image processing experts using a combination of top-down and bottom-up approaches, and used in a workflow context. This has suggested a shift in solving VIP tasks using multiple executables rather than single executables as typically practised by image processing experts. The illustration of the process models have shown that using this approach has enabled the derivation of multiple combinations of VIP solutions for the same task via the use of decision points, making it more flexible than previous approaches that can only derive a single sequential way for solving a VIP task. The next chapter will describe how these tools are utilised by the planner for workflow enactment.

Chapter 6

Workflow Enactor and Planner

At the heart of the system lies a workflow enactor that interfaces the interactions with the user and coordinates all the activities between the components within the system. The main component is a planner that is responsible for the derivation of video and image processing (VIP) solutions based on the provided goal and domain descriptions. Therefore, the planner is a reasoner that translates the high level non-technical terms (user goals and preferences) to low level technical terms (VIP operations) for workflow composition. This is done with the assistance of the process library and ontologies. Two key innovations of the planner are the ability to support workflow execution (interleaves planning with execution) and can perform in automatic or semi-automatic (interactive) mode. It extends the capabilities of typical planners by guiding users to construct more optimal solutions via the provision of recommended descriptions for the tools. It is also highly adaptable to user preferences. This chapter will provide the details of the workflow layer (Section 6.2), the workings of the planner (Section 6.3) and demonstrate two examples of the integration in the context of constraints and video descriptions (Section 6.5).

6.1 HTN Planners

Planning systems are useful for solving complex problems by generating action sequences for goals given the initial state of the world and a set of possible actions. Planning technology has been applied to a variety of applications, including spacecraft mission control, robotics, manufacturing, web and grid environments, emergency evacuation and games.

Traditional planners have existed for some 40 years now, a few representative clas-

sical planners include STRIPS [34], GraphPlan [10] and FF [46]. In classical AI planning, the planner will have to perform a trial-and-error search to find choices of actions to take based on their declarative descriptions and capabilities. This makes the search space immense although the use of heuristics can help make the search more efficient. One way to overcome this problem is by providing prescriptions on how to perform complex tasks within the domain model of the planner. This can be achieved using hierarchical task network (HTN) planning, initially proposed by Sacerdoti in NOAH [96] and Tate in NONLIN [103].

HTN planning uses so-called methods or refinements to describe the decomposition of non primitive tasks in terms of primitive and other non primitive tasks. Primitive tasks are directly executable by applying an operator. The planner starts with an abstract network representing the task to be accomplished, and proceeds by expanding the abstract network into more detailed networks lower down in the abstraction hierarchy, until the networks only contain executable actions. HTN methods are a form of domain-specific knowledge (in the form of task decompositions). This greatly reduces the search space of the planner by encoding knowledge on how to go about looking for a plan in a domain and also enables the user to control the type of solutions that are considered. Due to its successes and practicalities, several major HTN planners have been developed over the past decades, several of which are described next.

O-Plan [26] is a domain-independent general planning and control framework with the ability to employ detailed knowledge of the domain. The system combines HTN planning, agenda-based control architecture for efficiency, least commitment approach, temporal and resource constraint handling using incremental algorithms and opportunistic selection of the focus of attention on each problem solving cycle. It was aimed at applications involving project management, distribution logistics and space probes, amongst others. Its more recent version, O-Plan2 [85], has an agenda driven blackboard architecture in which each control cycle can post pending tasks during plan generation. Different knowledge sources with different skills are responsible for processing the pending tasks from the agenda. O-Plan2 manipulates a plan state which is a data structure containing the emerging plan, the flaws remaining in it and the information used in building the plan. O-Plan2 guarantees to produce at least one valid solution to a given problem if this is feasible within the constraints specified on the task and within the modelling capabilities provided by the constraint managers installed. In order to alleviate search, O-Plan2 envisages the implementation of constraint propagation mechanisms, in particular for temporal reasoning and resource reasoning. How-

ever, O-Plan2 does not guarantee to produce more than one such valid solution for any problem. O-Plan2 is not suitable for problems in which all (syntactically different) solutions are required or in which an optimal solution is needed.

I-X [104] is a portable cross-platform Java-based planning and collaboration support environment. It integrates several aspects such as representation and reasoning (via ontologies), user interfaces and applications (which include a planner based on O-Plan) in order to achieve an open, flexible and embeddable system. It is motivated by the need to provide a human-friendly environment to assist in the knowledge acquisition and knowledge management processes. Its planner, I-Plan, can perform hierarchical partial-order composition of plans and can propose alternative ways in which activities can be expanded. From there, the user can choose to proceed with a plan, check the validity of this plan, or replan. However, this must be done via trial-and-error cycles. What could help make this less tedious for the user is by providing descriptions for each option that could help the user to make more informed decisions.

SIPE-2 [117] is a domain-independent, hierarchical, and nonlinear AI generative planning system with mechanisms for reasoning about context-dependent-effects and solving practical problems. It interleaves planning with execution and has some replanning capabilities. It is a component of CYPRESS, a system that supports planning capabilities including action specification, generative planning, reasoning about uncertainty, reactive plan execution, and dynamic replanning. Among SIPE-2's features include interactiveness with user and supports multi-agent planning. It has a graphical interface that allows users to interact with the planner. Similar to O-Plan2, SIPE-2 implements a depth-first search, with chronological backtracking. However, it uses heuristics to guide search which means that it does not guarantee that a solution will be found, if it exists. SIPE-2 has been used to solve a range of problems, such as air campaign planning, military operations planning, oil spill response, production line scheduling, construction planning, robotics and toy problems and puzzles.

SHOP [80] and SHOP2 [79] are domain-independent HTN planners based on ordered task decomposition, where tasks are planned in the same order that they will later be executed. This avoids some goal-interaction issues that arise in other HTN planners, so that the planning algorithm is relatively simple. SHOP plans for tasks that are totally ordered, SHOP2 extends this capability to deal with partially ordered plans. Since SHOP2 can interleave subtasks of different tasks, some planning domains can be represented more easily in SHOP2 than in SHOP. These planners have been applied to problems that include logistics and blocks, transportation, space, games and numeric

domains and remain the most prominent HTN planners to-date.

HyHTN [69] is a hybrid planner that combines state-advancing HTN reduction strategy for hierarchical refinements, and a fast forward search for goal (pre-condition) achievement. Thus the performance of *e.g.* SHOP-like algorithms in HTN planning, and the performance of fast forward algorithms in pre-condition planning have been combined into a hybrid system. It was part of an initiative that aimed to develop a platform that can deal with structurally complex domains, but also transparent and portable enough to be used for research and experimental use. It is fed with a statically-checked domain model whose methods are encased in pre- and post-conditions and engineered *a priori* via its GUI to conform to the transparency property. HyHTN is integrated into a GUI that creates and maintains hierarchical domain specifications, and verifies them using a structural property checker. HyHTN has been applied to problem tasks similar to those tested with SHOP.

Successes of existing HTN planners indicate that HTN planning has provided a reliable means for solving complex problems where "recipes" for how to perform tasks can be easily encoded to help reduce the search space of the planner. However, their application in the VIP field has not been explored much. This is due to the absence of domain modelling within VIP in order to allow the planners to work upon. Recognising the considerable knowledge engineering efforts involved in modelling a domain, current trends in HTN planning are focused on learning HTN domain models automatically. Some examples include ICARUS [56], HTN-maker [47], HTN-learner [119] and probabilistic HTNs (pHTNs) [59]. However, the extent of their applicability to complex domains, such as VIP, is still not known, or in preliminary stages. VIP is a challenging problem domain that involves complex numerical computations, but also other modelling primitives such as iterations, condition statements and domain-dependent heuristics in the selection of appropriate algorithms.

To tackle the problems in the VIP domain, some commonalities in how these tasks are solved by image processing experts are determined and constructed manually. A domain-independent planner fed with domain-specific heuristics can be tailored to solve VIP tasks. Furthermore, the incorporation of ontologies would prove beneficial and simplifies some of the reasoning utilities of the planner by reusing the knowledge inferred from them. This has been based on prior work provided in [74] and [76]. How this is achieved will be illustrated in the next section.

6.2 Workflow Enactor

Chapter 3 described the architecture of the workflow composition system devised for this thesis. Chapter 4 has described the high level goals in the form of VIP tasks, as well as the low level VIP tools, which were elaborated further in Chapter 5. The goals are mapped to process models (Section 5.6) that are contained in the process library as methods. The process models can be decomposed from high level process nodes to low level process leaf nodes. At the leaf node level, the processes are instantiated to VIP tools that can be executed. The deliberation process involved in transforming the high level goals to low level software tools is provided by the planner. The workflow enactor further manages the flow of processing between the system and the user. This section gives a clearer view of the interaction between the workflow enactor, the planner and the user. This is described diagrammatically in Fig. 6.1.

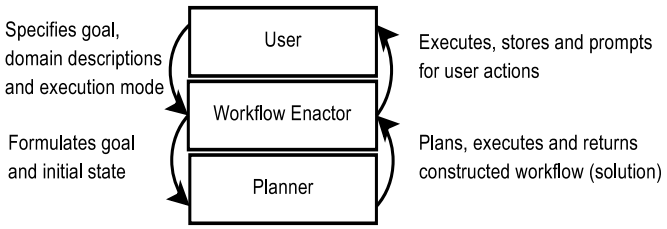


Figure 6.1: Communication between the workflow enactor, planner and user.

The workflow enactor sits in the unique position of communicating the user requests to the planner and *vice versa*. This delegation is done by switching control between the user and the planner that reasons with the help of the ontologies. It is also able to reexecute solutions provided by the planner. As the planner is equipped with execution capabilities (Section 6.3), the workflow stores this solution, which can be invoked when requested by the user or off-line. The workflow enactor is implemented using Prolog which is a high level, logical, declarative language useful for experimenting and prototyping AI algorithms. Prolog is also the selected choice for the planner and process library while the ontologies were described using FBPML-DL, a first ordered logical language that is compatible with Prolog (see Section 4.3.1). Hence, a declarative approach that would allow for more effective reasoning has been adopted. The features and functions of the workflow enactor are outlined next.

6.2.1 Features

There are various utilities that provide specific facilities that are accessed by the workflow components during different time points of the processing. The predicates and utilities are features that are maintained by the workflow. They are described below.

- **Input to planner.** Predicates that generate the goals, constraints, video description and state of the system. These predicates are asserted into the knowledge base and will be described in Section 6.3.
- **Domain information manipulation.** Predicates that modify the constraints and/or video descriptions. The predicates containing the original values are retracted and reasserted with the modified values.
- **Ontology utilities** are also contained in the workflow and accessed by the planner, workflow and process library. They are static. (Chapter 4 described these utilities in more detail for each ontology).
- **Process library utilities** such as the retrieval of video data information, *e.g.* directories and files containing video, frame information, background models and intermediate text and image results.
- The workflow also holds **general utility functions**, such as list manipulations and printing routines. These utilities are called by the planner, workflow and process library and are static.

6.2.2 Functions

The workflow enactor, being at a higher level of abstraction from the planner is responsible for the delegation of tasks between the system components and acts as the main mediator between the system and the user. The main roles of the workflow enactor include data and preliminary information capture (Section 6.2.2.1), goal formulation and planner invocation (Section 6.2.2.2) and passing back control to the user (Section 6.2.2.3). The main run of the workflow enactor consists of the following steps:

1. Retract existing domain predicates from previous run of workflow.
2. Read user request, capture input data, initial domain descriptions and planning mode (automatic or semi-automatic).

3. Formulate input for planner with the aid of goal and video description ontologies.

4. Open and initialise script file to store solution plans.

5. Invoke planner. Planning and execution is described in Section 6.4.

6. Display the solution plan, time taken to execute workflow, textual results and location of files containing final results.

7. Prompt user for next preferred step: i) Display the generated solution in video form. ii) Replan with modified parameters. iii) Select another goal. iv) Evaluate quality of result.

The main workings of the workflow system is presented in further detail next.

6.2.2.1 Data and Preliminary Information Capture

The workflow acts as the main moderator of events within the system. At startup, the workflow prompts the user for input. Fig. 6.2 illustrates the initial interaction between the system and the user. At the first instance the task they wish to perform is requested. This task is translated into the goal of the system.

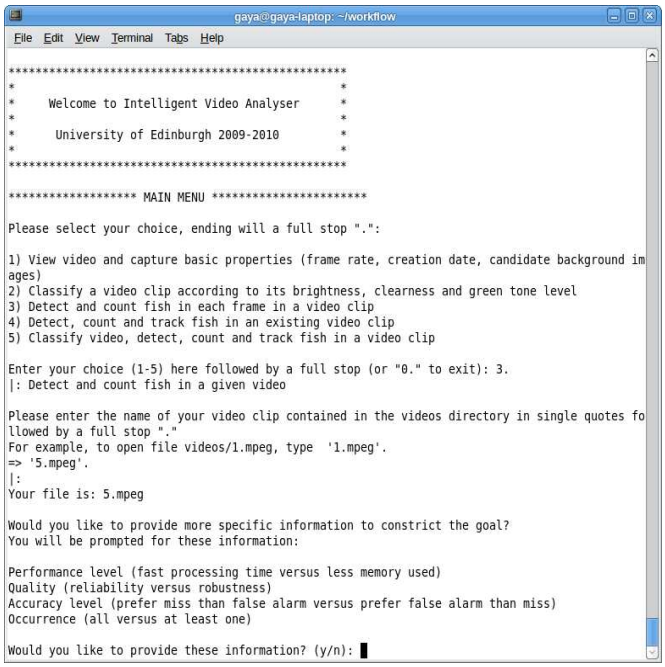


Figure 6.2: Welcome screen, user goal and input data capture.

After selecting the goal, the system further prompts the user for constraint information. As defined in Chapter 4, the constraint information include performance level, quality criterion, accuracy level and occurrence. Fig. 6.3 depicts this.

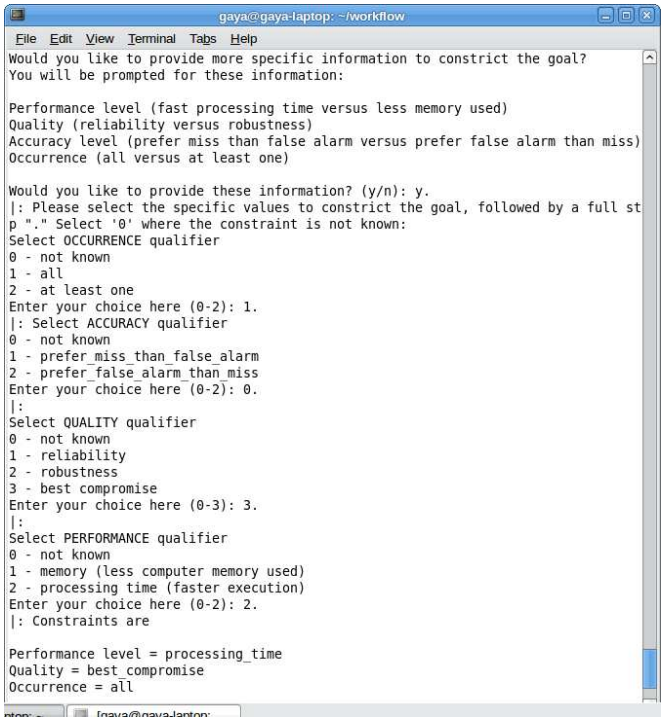


Figure 6.3: Constraints capture provided by user.

In all of these cases, the user may choose not to provide any constraints. In this scenario, default values are set for each constraint criterion. These are recommended by image processing experts as reasonable values for the workflow to proceed with. The following default values are relevant for the respective constraint criterion.

- Performance: processing time.
- Quality: best compromise.
- Accuracy: prefer miss than false alarm.
- Occurrence: all.

Next, the video descriptions are requested from the user. These include brightness, clearness and green tone (algal) levels. If the video descriptions are not entered by the user, their initial values are set to be ‘not known’. At a later stage, when the video is being pre-processed, the video descriptions are computed automatically during

preliminary analysis (see Section 5.5.1). Hence there is an alternative mechanism that deals with the retrieval of domain information in the absence of user input.

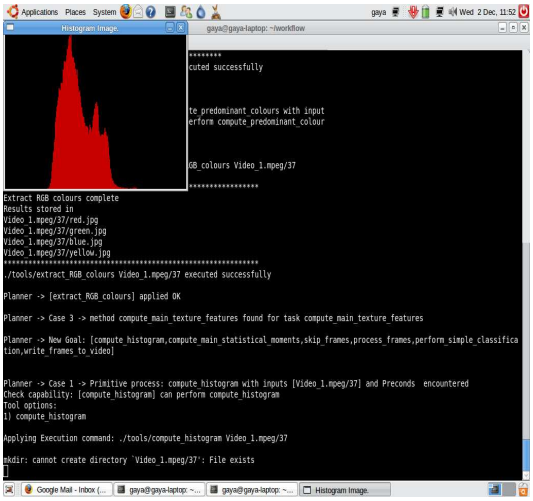
Apart from data-related information, the mode in which the solution is derived by the planner is also determined by the user. Two modes are available, automatic and semi-automatic. In the automatic mode, the reasoning is done purely without any intervention from the user. So the user will be presented with a solution based on the system's reasoning alone. In the semi-automatic mode, the user will be prompted to make selection of planning operators in the form of VIP tools. The advantage of having this option is it enables the construction of more optimal solutions based on user feedback. This mechanism is described further in Section 6.3.

6.2.2.2 Goal Formulation and Planner Invocation

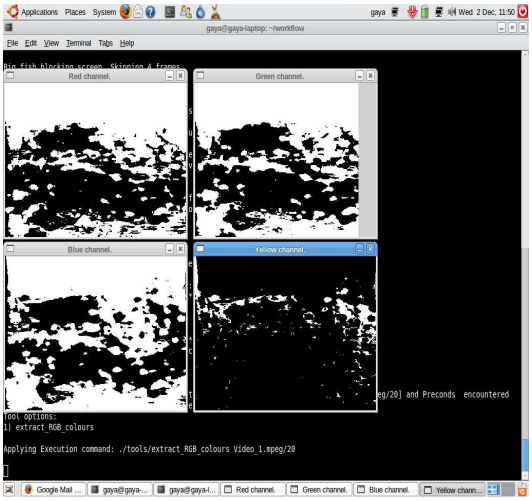
Using the information acquired from the user, default values or automatically generated values, the goal, initial state and domain information are formulated. This is achieved with the assistance of the goal and video description ontologies. Section 4.8 describes this process in more detail. The goal tasks are represented in a list while the domain information and initial state are contained in predicates that are asserted (see Section 6.3). Having the goal tasks in a list would ease the workings of the planner that follows a decomposition-based approach that views the goal as a list of tasks to be done. Once the goal, initial state and domain information are formulated, they are passed as input to the planner. Planning will take place as long as the goal and input video are given. The execution mode is also passed into the planner so that it knows if user intervention is required or not.

6.2.2.3 Passing Control Back to User

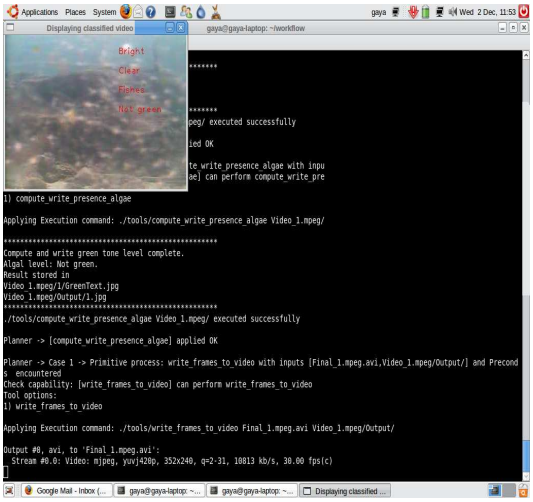
As the planner adopts a planning interleaved with execution scheme, the workflow translates the generated plan into an executable script which can be reinvoked. Once the planning and execution of a task is complete, the user has the choice to manipulate the domain descriptions. In this case replanning will take place. The user may also select to start all over (*e.g.* to choose a new goal) or evaluate the optimality of the result. Fig. 6.4 shows a series of screen shots to illustrate the workings of the system to reflect this for a video classification task. The next section goes into details of the deliberation process that takes place within the system, *i.e.* the planner.



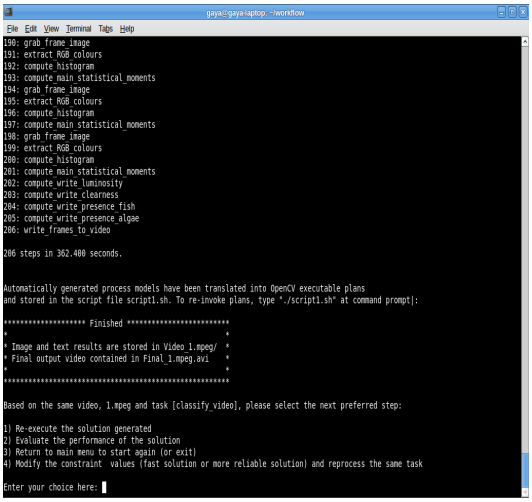
(a) Compute histogram – Visual and text display.



(b) Compute RGB channels – Visual results displayed to user.



(c) Final result – Classified video file displayed to user.



(d) End of planning and execution – Text results and location of files containing the final results are displayed. User is prompted for the next step.

Figure 6.4: Snapshots of workflow enactment for video classification task.

6.3 Planner Design

The planner was designed with the aim of enhancing the capabilities of existing HTN planners. Among the features that have been added include interleaving planning with execution, planning in automatic and semi-automatic (interactive) modes and enabling the user to construct more optimal solutions by providing recommended descriptions when more than one tool is available to perform a primitive task.

The implementation of the planner has been kept separate to the modelling of the domain. The planner itself is domain-independent and can be tailored to be used in different problem scenarios as long as the domain descriptions and relevant ontologies are provided. Domain-specific information is encoded in the knowledge base as facts and in the process library as primitive tasks (Section 6.3.4) and methods (Section 6.3.5). The planner and domain model are described declaratively using SICStus Prolog 4.0. The planner was built based on ordered task decomposition where the planning algorithm always builds plans forward from the initial state. This eases representation as ordered task decomposition which composes tasks in the same order that they will be executed is analogous to Prolog's search strategy.

The input to the planner are the goals, objects and conditions of the objects at the beginning of the problem (initial state), and a representation of the actions that can be applied directly to primitive tasks (operators). For HTN planning, a set of *methods* that describe how non primitive tasks are decomposed into primitive and non primitive tasks are also required. The output should be (partial) orderings of operators guaranteed to achieve the goals when applied to the initial state.

A video processing task can be modelled as an HTN planning problem, where a goal list, G is represented as the VIP task(s) to be solved, the primitive tasks, p are represented by the VIP primitive tasks and the operators, O are represented by the VIP tools that may perform the primitive tasks directly. The methods, M specify how the non primitive tasks are decomposed into primitive and non primitive subtasks. The primitive tasks and methods are contained in the process library.

6.3.1 Preliminaries

Adapting the conventions provided in Ghallab *et al.* [39], an HTN planning problem is a 5-tuple

$$P = (s_0, G, P, O, M)$$

where s_0 is the initial state, G is the goal list, P is a set of primitive tasks, O is a set of operators, and M is a set of HTN methods. A primitive task, $p \in P$ is a 3-tuple

$$p = (\text{name}(p), \text{preconditions}(p), \text{postconditions}(p))$$

where name(p) is a unique name for the primitive task, preconditions(p) is a set of literals that must hold for the task to be applied and postconditions(p) is a set of literals that must hold after the task is applied.

An HTN method, $m \in M$ is a 6-tuple

$$m = (\text{name}(m), \text{task}(m), \text{preconditions}(m), \text{decomposition}(m), \text{effects}(m), \text{postconditions}(m))$$

where name(m) is a unique name for the method, task(m) is a non primitive task, preconditions(m) is a set of literals that must hold for the task to be applied, decomposition(m) is a set of primitive or non primitive tasks that m can be decomposed into, effects(m) is a set of literals to be asserted after the method is applied and postconditions(m) is a set of literals that must hold after the task is applied. The planning domain, D, is the pair (O, M).

6.3.2 Initial State

Planning involves the representation of actions and world models, reasoning about the effects of actions, and techniques for efficiently searching the space of possible plans. A state represents a situation or snapshot of an application. This is sometimes called a system state or world state.

The system's state is represented by predicates that hold to depict the snapshot of the world at a certain time point, called facts. These facts constitute the domain of the world and the objects that exist. At the start of the system, the initial domain conditions, or *initial state*, are determined. The facts that are directly related to the domain of video processing include (but not restricted to) the following:

- Video input file (predicate input_file/1).
- Current frame number (predicate frame_no/1).
- Total number of frames (predicate num_frames/1).

The facts that are not related to the domain but needed for storage of results include

- List containing the resulting steps (predicate sol_list/1).
- File to store the executable plan (predicate script_file/1).

It is intuitive that the current frame number when the processing starts is 1. However, the total number of frames in a video will need to be computed from the video itself and can only be determined once the video's name is specified by the user. This will be the same for all the predicates, hence they will be asserted dynamically once determined.

6.3.3 Domain Modelling

Modelling the domain for VIP tasks can be seen as a dynamic process, as with the initial state. The domain model for VIP tasks consists of the following:

- Constraints (performance, quality, accuracy and occurrence levels). See Section 4.4 for a detailed explanation on these.
- Video descriptions (*e.g.* brightness, clearness and green tone levels). See Section 4.5 for a detailed explanation of these.

These values are computed during the domain acquisition phase, either from the user or computed automatically. Other domain information that are determined throughout the planning process include speed of movement, noise type, presence of fish blocking screen, green tone level, variance, skewness, uniformity, entropy, background movement and background model. The predicates are asserted via an *add list* facility, encoded as effects after the application of an operator. The planner algorithm in Section 6.4 will describe this mechanism further.

6.3.4 Primitive Tasks and Operators Representation

Chapter 5 described the 30 operators identified and developed for the VIP task to perform video classification, fish detection and counting. The corresponding primitive tasks that these operators may act upon are encoded in the process library. As stated earlier, primitive tasks are those that can be performed directly by operators or VIP tools. Primitive tasks are represented by the predicate independent_executable/6. Specifically this predicate is defined as follows:

independent_executable(User_Terminology, Function_Call_Name, Preconditions_List, Effects_File, Input_List, Output_List).

This predicate contains the name of the primitive task, User_Terminology, its corresponding technical name, Function_Call_Name, its preconditions, Preconditions_List, the parameter values, Input_List, output values, Output_List and the file that contains the predicates that are asserted as a result of applying this task, Effects_File. The typical values for the parameters are the video name, the frame and video directories, current frame image, background model image and background model directory. Appendix D gives a more thorough treatment of these terminologies.

The list of preconditions, effects and postconditions of the primitive tasks are assumed to be a conjunction of literals. For the preconditions, they are all the conditions that must hold (prerequisites) for this primitive task to be performed. The effects are conditions that will be asserted or retracted after completion of the task. For the postconditions, they are all the conditions that must hold *after* the task is applied. Since the effects are often computed during the execution of a VIP operator, the predicates that need to be introduced or removed are stored in a text file specified by Effects_File.

The operator(s) that may perform the primitive task is contained in the capability ontology (described in Chapter 4). Technically, a ground operator O can be applied to a state S if all the preconditions of O are satisfied in S. If operator O is applied to state S then the new state as a result of this application $O(S)$ would be the current state S plus the add list (contained in Effects_File) of O. Another assumption is that a primitive task satisfies the rule of default persistence, that is if a primitive task's specification does not specify that a fact changes, then it does not change.

6.3.5 Non Primitive Tasks – Methods

Non primitive tasks are decomposable to primitive and non primitive subtasks. Schemes for reducing them are encoded as *methods* in the process library. The predicate that holds information for non primitive tasks is method/5. For each method, the name of the method, the preconditions, decomposition, effects and postconditions are specified. The preconditions and postconditions are modelled as a list of conditions that must all hold before and after the application of the subtasks in the method respectively, similar to the modelling of preconditions and postconditions for primitive tasks. The decomposition is given by a set of subtasks that must be performed in order to

achieve this non primitive task. The effects are a list of predicates that will be invoked (add and delete lists) as a result of applying the method. The invocation of the effects is achieved by utilising the built-in Prolog predicate call/1. In principle this is the same as the assertion of predicates in the add list for primitive tasks, except that the add lists for methods are not computed by previous VIP operator invocations, thus not needing to be stored in a file, but rather encoded in the method directly.

For VIP tasks, the methods are broadly categorised into three distinct types; non recursive, recursive and multiple conditions.

6.3.5.1 Non Recursive Methods

This is the most common form of method that has a list of preconditions, decomposition, effects and postconditions. An example is the method for the primitive task classify_video that classifies the video according to its brightness, clearness and green tone levels. Its formal description is given below.

```
% classify video
method(classify_video, [], [process_frames, perform_simple_classification, write_frames_
to_video], (), []).
```

This predicate can be read as follows: The non primitive task, 'classify_video' can be achieved by performing the subtasks 'process_frames', 'perform_simple_classification' and 'write_frames_to_video' in a sequence. There are no preconditions, effects and postconditions associated with this method. Note that 'process_frames', 'perform_simple_classification' and 'write_frames_to_video' can be primitive or non primitive, if non primitive, they will be specified as methods as well.

6.3.5.2 Recursive Methods

Loops are encoded within the method construct using recursion, as follows. For a recursive non primitive task, its decomposition consists of a list of its subtasks and itself. This way, a call to itself is performed after all its subtasks are performed, creating a loop. The base case will have an empty list, when appended to the start of the goal list, the goal list will now contain the next goal to be processed and the loop is thus terminated. In this way loops can be handled efficiently.

A typical example of loop usage is to process a sequence of subtasks for each frame

in a video. In a video classification task, for each frame, the frame image is acquired, followed by the computation of its colour and texture features. This is represented in the recursive method for the subtask 'process_frames'.

```
% process_frames: recursive case
method(process_frames, [X =< Y], [grab_frame_image, compute_predominant_colours,
compute_main_ texture_features, skip_frames, process_frames], (nl), []) :-
    frame_no(X),
    total_frames(Y).

% process_frames: base case
method(process_frames, [X > Y], [], (nl), []) :-
    frame_no(X),
    total_frames(Y).
```

Note that the base case of the method will be reached when the current frame number exceeds the total number of frames in the video. This is controlled via the precondition [X > Y], where X is the current frame number and Y is the total number of frames.

6.3.5.3 Multiple Conditions

This type of method is similar to the non recursive method, the difference being in the definition of multiple cases for the same method based on different preconditions that would result in different effects. The decomposition itself does not contain any task and therefore does not require the application of any VIP operator, only a state transition takes place. An example is the method to describe the subtask skip_frames that is responsible for determining the next frame in a video to process. In an ordinary sense, the next frame to process would be the one that immediately follows the current frame. However, if the screen is blocked by an object, *e.g.* a fish, then the program can safely skip four frames and proceed with processing from the fourth frame after the current frame. Another condition that affects this is the constraint criteria. Should the user specify that a fast algorithm is required, *i.e.* the performance criterion is set to processing time, then the system does not need to process all the frames and can skip half the total number of frames to the middle of the video. The three cases that apply to the method skip_frames are shown below in Prolog.

% Case 1: Screen is blocked, skip four frames
method(skip_frames, [blocked(1)], [], ((frame_no(X), X1 is X+4, retract(frame_no(X)), assert(frame_no(X1)))), []).

% Case 2: Fast processing is required, skip to middle of video
method(skip_frames, [performance(processing_time)], [], ((frame_no(X), Half is X/2, X1 is X+Half, retract(frame_no(X)), assert(frame_no(X1)))), []).

% Case 3: Normal case, proceed to next frame
method(skip_frames, [blocked(0)], [], ((frame_no(X), X1 is X+1, retract(frame_no(X)), assert(frame_no(X1)))), []).

The ordering of the different cases for this method plays a vital role in the sequence of execution. The first case will be identified first if its precondition holds. This implies that even if other criteria, such as the processing time, holds (Case 2), this case is given priority over them. If this method is not applicable (*i.e.* the screen is not blocked), then the next case is taken into consideration. Hence methods with multiple conditions are ordered according to the level of priorities - the method that should be considered most is described first, and so on. By encoding these different types of modelling primitives within the method constructs, a rich representation for the VIP domain can be achieved. A complete list of the methods is provided in Appendix D.

6.4 Planner Algorithm

Planning algorithms solve problems by applying *applicable* operators to initial state to create new states, then repeating this process from the new states until the goal state is reached. In HTN planning, the algorithm stops when all the goal tasks have been achieved. The algorithm below describes the main workings of the planner implemented for this thesis.

```
gplanner(initial-state S, goal-list [g1|G], domain D, solution-plan P)
Initialise P to be empty
If goal-list is empty, return P

Consider first element in the goal-list, g1 in goal list [g1|T]
```

```
Case 1: If g1 is primitive AND all its preconditions hold
       1.1. If one or more operator instances (tools) match g1
            Retrieve ALL operators (tools) T = [t1,..,tn] from capability
            ontology that can perform g1
            For each tool, ti in T
               Retrieve suitable domain conditions for ti from capability ontology
            End For
              If more than one operator is available to perform g1
                Display all applicable tools with suitable domain conditions
                Prompt user to select preferred tool, tp
              Else tp is the only available operator to perform g1
                Apply tp to S and planning algorithm to rest of goal-list:
                apply-operator(tp, Input_list, Add_list)
                Check that all postconditions of g1 hold
                gplanner(tp(S), G, D, P)
       1.2 Else return fail

Case 2: If g1 is non primitive
       2.1. If a method instance m matches g1 in S
            AND all its preconditions hold
              Append the decompositions of m into the front of the goal-list
              Add all elements in m's Add List to S
              Check that all postconditions of m hold
              Apply planning algorithm to this new list of goals:
              gplanner(S, append(m(g1),G), D, P)
       2.2 Else return fail

% Apply_operator
apply-operator(Tool, Input_list, Add_list, P, S)
    Update solution-plan P with Tool (append Tool to the end of P)
    Execute Tool with parameters Input_list
    Add all elements in Add_list (effects) to S
```

Algorithm 2: Workings of the enhanced HTN planner in semi-automatic mode.

The domain is represented by the predicates that contain video descriptions (*e.g.* brightness, clearness and green tone levels), the constraints (*e.g.* accuracy and processing time), methods (decompositions) and operators (tools to execute the primitive tasks). The algorithm is a recursive function that works on the goal list until it is empty. It inspects each item in the goal list to see if it is a primitive task. If the item is a primitive task, it seeks to find an operator that can perform the primitive task. This is done

automatically or semi-automatically, depending on the planning mode selected by the user. Once found, the operator is applied and the primitive task is accomplished. If the task is not primitive, it looks for a method instance that matches it and appends its decompositions to the start of the goal list. The basis for the planning algorithm was taken from HTN planners that generate plans in a totally ordered fashion, *e.g.* SHOP [80]. Tasks are decomposed from left to right in the same order that they will be executed. In addition, it can plan interactively, interleave planning with workflow execution and has been enriched to make use of knowledge from ontologies.

6.4.1 Automatic or Interactive Mode of Planning

The planner works in either automatic or semi-automatic (interactive) mode, determined by the user before planning takes place. In the automatic mode, user intervention during tool selection is not required. At each planning step, the planner itself selects the tool deemed most optimal based on the domain conditions that match with the tool's recommended domain conditions encoded in the capability ontology. These conditions have been determined based on IP experts' heuristics. Thus the preferred step in Algorithm 2 is selected by the system rather than the user. When there are no heuristics to guide the tool selection, the first tool encountered that can perform the primitive task is selected for execution. Hence it follows a depth-first style of search.

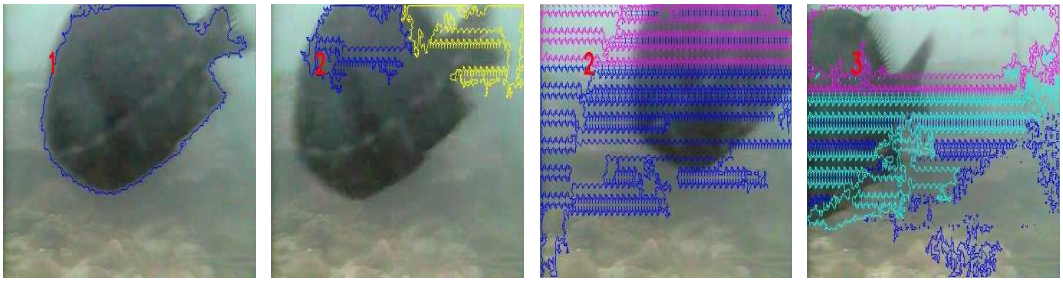
In the semi-automatic mode, the planning process in **interactive** when more than one tool is available to perform a primitive task. At this level, the planner derives all the applicable VIP tools and their recommended domain descriptions from the capability ontology (See Algorithm 1 in Section 4.8 for the reasoning mechanism of this). The user selects the VIP operator/tool of their choice based on the recommended domain descriptions for each tool as guidance. Thus the planner allows the user to select a tool during the planning process, giving them control and also the ability to make informed decisions. The next section will highlight how this control is followed through with user verification of the final result.

6.4.2 Interleaving Planning with Execution

The planner follows a style of planning interleaved with execution. Once a VIP tool is selected, it is executed before the next planning step is inferred. This is because domain conditions could change as a result of the application of a selected tool. This would then affect the planning process of subsequent steps. However, replanning is

not allowed until execution of the whole task is completed. This is due to the fact that intermediate results cannot be used to assess the optimality of the selected VIP tools (neither by the system nor the image processing-naive user) and finding a suitable heuristic for this purpose is not trivial.

Once planning is complete, the user has access to the final video containing the result of applying the tool(s) that they have selected. This will give them a good indication of the optimality of the tool(s) that they have selected. Fig. 6.5 shows an example of this for a detection task. After viewing this result, they may decide to replan in order to try different choices of tools. Chapter 7 includes an evaluation of the learnability level achieved by the user in selecting the optimal solutions based on the descriptions provided by the system using the semi-automatic planning mode. The next section will illustrate two examples on how domain descriptions can affect the selection of planning operators.



(a) Adaptive Gaussian Mixture Model. (b) IFD Model. (c) Poisson Model. (d) W4 Model.

Figure 6.5: Results of applying four different background models for fish detection and counting task for the same video.

6.5 Utilising Domain Descriptions for Tool Selection and Planning

This section illustrates the influence of domain descriptions (constraints and video descriptions) on the planning process. The examples are drawn from two separate scenarios within the tasks mentioned in Section 6.2.2.1. The first concerns the usage of constraints for video classification whilst the second concerns the usage of constraints and video descriptions for object detection.

6.5.1 Constraints Influencing Speed of Processing

In the selection of steps for video classification task, the combination of constraint criteria Performance and Quality determine whether a fast classification that considers only a fraction of the frames is performed on the input video or a slower processing that takes all the frames into account is performed. When the user selects this task, they are prompted if they prefer a fast (but possibly less reliable) solution or a more reliable (but slower) solution. If they select the former, the Performance criterion is set to hold the value processing_time, otherwise the Quality criterion is set to hold the value reliability if they select the latter.

Planning starts with the first frame of the video, when it finishes processing the first frame, it encounters the method skip_frames, which determines the next frame to proceed to. As explained in Section 6.3.5.3, the predicates that contain this are encoded as methods. If the precondition 'Performance = processing_time' holds, then half the number of frames will be skipped. As constraint values are static and do not change during the planning process, this will repeat until the last frame encountered exceeds the total number of frames in the video.

6.5.2 Domain Descriptions Influencing Background Model Algorithm

Background models play an integral part in tasks that involve object detection and tracking. A background model is a representation of the video without any foreground objects. Usually this is in the form of an image. For a video processing task that involves fish detection, seven valid types of background models have been made available for this thesis, they are summarised in Section 5.5. These include adaptive models such as Adaptive Gaussian Mixture model [120] and non adaptive models such as Intra-Frame Difference (IFD) model [8]. However, the suitability of a background model to be used on a particular video would depend on the present domain conditions, such as the clearness level, level of background movement and percentage background object, among others. It would also depend on the constraint criteria Accuracy and Performance. Based on image processing experts' heuristics, four video description criteria and two constraint criteria have been identified as influencing factors for the background models algorithms.

The table in Fig. 6.6 contains these criteria and their appropriate values for each background model. Each video description may have three values ("low", "medium",

Background Model	Video Description				Constraint	
	Uniformity	Clearness	Background Movement Speed	Percentage Back-ground Object	Accuracy	Perfor-mance
Adaptive Gaussian Mixture (48)	low medium	low medium	high	–	–	–
Adaptive Poisson (24)	high	low medium	high	–	–	–
Moving Average (18)	high	–	low	–	prefer false alarm than miss	–
Poisson (18)	high	–	low	–	–	processing time
W4 (36)	low medium	–	high	–	prefer miss than false alarm	–
Gaussian (36)	–	high	high	–	–	–
Intra-Frame Difference (48)	low medium	–	low medium	high	–	–

Figure 6.6: Suitable Domain Descriptions for Creating Background Model Algorithms.

"high") and each constraint criterion may have two values. Chapter 4 described the constraints and video descriptions in more detail. The number in brackets beside each background model indicates the number of cases covered by the combination of domain description values for that model. Dashed entries (–) denote that the background model is not affected by the corresponding domain description.

These criteria account for 228 cases or 70.4% out of the total 324 possible combinations of the six domain descriptions listed above. The breakdown by background model type includes Adaptive Gaussian Mixture 14.8%, Adaptive Poisson 7.4%, Moving Average 5.6%, Poisson 5.6%, W4 11.1%, Gaussian 11.1% and Intra-Frame Difference 14.8%. The remaining 30% will be catered by the usage of Gaussian Mixture model fused with Moving Average model [99]. This is also explained in Section 5.2.

Initially, for each background model type, the conjunction of these domain description values were incorporated as preconditions in the methods for the subtask 'create_background_model'. When planning is done automatically without user intervention, the planner will choose the background model that fits all the domain descriptions for the video, otherwise it will create a background model from the combination of Adaptive Gaussian Mixture and Moving Average models. However, encoding such preconditions is too specific or restrictive, often resulting in none of the background models to be selected. Therefore, they are encoded as recommended domain descriptions within the capability ontology instead. Section 4.8 explains how the ontologies are used in the deliberation process.

When planning in semi-automatic mode, the user is presented with all the available background models, they are also given the recommended domain descriptions for each background model to assist them with the decision-making process. The advantage of planning in the semi-automatic mode is that users can verify the completeness and optimality of the plans that they have selected. The final video, which is the result of the execution of the workflow resulting from the application of all the planning steps, will display visually to the user whether the goal has been achieved (completeness), and the accuracy of the detection (optimality). Hence, if the result is not satisfactory, the user can choose to replan the same task but by selecting a different background model. Over time, the user can 'learn' which background model tool will work best for a given type of video. Finally the user can assess the quality of a particular solution by giving a performance measure for it. The system can use this information to learn optimal plans. However, the system's learnability is outside the scope of this thesis.

6.6 Extensibility of Workflow

As the workflow is an integration of several techniques, the level of extensibility should be taken into consideration. This section illustrates the extensibility of the hybrid workflow composition and execution framework by demonstrating the engineering efforts involved in adding and removing a planning step (VIP tool).

6.6.1 Cost of Adding a Planning Step

Suppose a new VIP tool is available to perform a primitive process. First this tool will need to be added into the capability ontology, as an instance and tied to a VIP operation (via the function 'canPerform'). Then, it will need to be represented in the process library as an independent executable via the predicate 'independent_executable/6'. Assuming that the primitive step that it can perform is one that already exists, no other engineering work is required. This takes three lines of Prolog code and can be done in three minutes. If the performance cost for this tool is known, an extra line of code via the predicate 'hasPerformanceIndicator' will need to be added. No changes to the planner or workflow code is required.

6.6.2 Cost of Removing a Planning Step

Conversely, when a tool is no longer available or has been deemed unsuitable to perform a primitive task, it should be removed from the process library and capability ontology. This requires the removal of the same code as for the addition of a tool (three lines of code). However, the tool can remain within the system and only be disabled from performing the primitive task by removing the predicate 'canPerform'. In this way, the tool will not be discovered during the planning step.

The removal of a planning operator would not affect the running of the planner and overall system as long as there is at least another tool that is able to perform the same task. If it happens to be the only tool that can accomplish the designated task, then the planning process would fail if the solution plan happens to include this task. Therefore, while the cost of removing a planning step is fairly straightforward, care should be taken to ensure that it retains the validity of the solutions produced after the removal of this step. Rigorous testing should be carried out to validate the planner's and workflow's behaviours using suitable test cases. These could include white-box and black-box testing approaches in software engineering.

The cost of adding a new method is trivial as it involves the addition of one line of code via the predicate ‘method/5’. As can be seen, the workflow framework is easily extensible to include new tools with minimal engineering effort. Section 7.6 will illustrate how this integrated approach using multiple executables is more adaptable than single execution approaches typically undertaken by image processing experts.

6.7 Planner Illustration

In this section, the workings of the planner is illustrated by emphasising choices of paths that can be taken for the VIP tasks determined for this thesis. For this purpose, the main methods in the process library are illustrated here as subsections. There can be several ways a method can be performed, shown by the numbered cases in each subsection. Sample plan traces and analysis are provided where possible. Primitive tasks and methods (non primitive tasks) are distinguished by the inclusion of the clause `(m)` at the end of methods. For each method, preconditions are encoded in square brackets, `[]`, expressed in natural language and represented as a conjunction of literals (comma-separated). Process models from Section 5.6 are referenced where relevant.

6.7.1 Goals

When planning begins, the top goal or high level VIP task is taken as a starting point. For the thesis, five goals have been determined. They are described in detail in Section 7.3. The paths taken by the different goals are as follows:

```
1. [goal=classify video]: process_frames (m) --> perform_video_classification (m)
   --> write_frames_to_output_video
2. [goal=detect, count fish]: preliminary_analysis --> process_frames (m) -->
   write_frames_to_output_video
3. [goal=detect, count and track fish]: preliminary_analysis -->
   process_frames (m) --> write_frames_to_output_video
4. [goal=classify video, detect, count fish]: preliminary_analysis -->
   process_frames (m) --> perform_video_classification (m) -->
   write_frames_to_output_video
5. [goal=classify video, detect, count and track fish]: preliminary_analysis -->
   process_frames (m) --> perform_video_classification (m) -->
   write_frames_to_output_video
```

6.7.2 Process Frames

This method constitutes the main loop of a VIP task and is applied to (all) the frames in a video. The paths taken are dependent on the VIP goal, as illustrated below. Fig. 5.9 contains the visual representation of the path taken for the task `classify video` (Item 1). The base and recursive cases of this method are shown in Section 6.3.5.2.

1. [goal=classify video]: grab_frame_image --> extract_rgb --> compute_texture_features (m) --> skip_frames (m)
2. [goal=detect count fish]: grab_frame_image --> extract_rgb --> compute_texture_features (m) --> perform_detection (m) --> count_fish_frame (m) --> skip_frames (m)
3. [goal=detect track fish]: grab_frame_image --> extract_rgb --> compute_texture_features (m) --> perform_detection (m) --> perform_tracking (m) --> skip_frames (m)
4. [goal=classify video, detect count fish]: grab_frame_image --> extract_rgb --> compute_texture_features (m) --> perform_detection (m) --> count_fish_frame (m) --> skip_frames (m)
5. [goal=classify video, detect, count and track fish]: grab_frame_image --> extract_rgb --> compute_texture_features (m) --> perform_detection (m) --> perform_tracking (m) --> skip_frames (m)

6.7.3 Compute Texture Features

Texture features are used for all the processing of the tasks identified for this thesis. Two ways these features can be computed are given below. Instead of having any preconditions tied to these, they are just selected in order of preference. The first method that computes the histogram followed by the main statistical moments is given preference. If this step fails, then the Gabor filter algorithm is selected. Fig. 5.13 illustrates this selection visually.

1. compute_histogram --> compute_statistical_moments
2. compute_gabor_filter

6.7.4 Perform Detection

The paths taken for detection depend on the current frame number and also the type of background model. The background model types and their criteria for selection are explained in Section 6.5.2. The process model for this method is contained in Fig. 5.14.

1. [frame no=1]: create_bg_model --> detect objects and blobs (m)
2. [frame no>1, background model=moving average]: update_moving_average --> detect_moving_objects --> detect_correct_blobs

```
3. [frame no>1, background model=non adaptive]: create_bg_model -->
   detect_moving_objects --> detect_correct_blobs
4. [frame no>1, background model=adaptive]: (do nothing)
```

6.7.5 Detect Objects and Blobs

This method also depends on the type of background model. Adaptive models are created, applied and detect objects and blobs once for a video. Non adaptive models will need to be updated at each subsequent frame and thus the detection of blobs is also done iteratively over all the frames.

```
1. [bg model=non-adaptive]: detect_moving_objects --> detect_correct_blobs
2. [bg model=adaptive]: (do nothing)
```

6.7.6 Fish Counting

There are two ways fish counting can be done. One is by counting the occurrence of fish in each frame independently of other frames, *i.e.* without keeping track of fish from previous frames. The other is by performing tracking to take into account fish from previous frames. This task can keep count of the number of fish in the video so far. The two methods are presented below, and tied to the goal they are trying to achieve. They are selected during Process Frames (Section 6.7.2).

```
1. [count_fish_frame]: compute_connected_components --> compute_area_convex_hull_
   over_blob --> compute_camshift --> compute_write_num_fish_in_frame_and_video
   --> determine_fish_blocking_screen
2. [perform_tracking]: extract_hsv --> compute_backprojection -->
   compute_connected_components --> compute_area_convex_hull_over_blob -->
   compute_camshift --> compute_closest_blob (loop) --> compute_write_num_fish_in_
   frame_and_video --> determine_fish_blocking_screen
```

6.7.7 Skip Frames

As mentioned in Sections 6.3.5.3 and 6.5.1, this execution control construct deals with the next frame to process. The implications of skipping to different frames will have effect on the speed of the processing, as well as the accuracy of the final result. In order to provide more efficient processing, there are two ways to speed the processing of a VIP task. One is simply by not taking all the frames into account (Case 1). This can be applied securely for video classification tasks for the test data as it has been empirically tested that taking just three frames is sufficient for the classification of brightness, clearness and green tone levels (Section 7.5.5). Another way of speeding

up the processing is by ignoring frames that contain a big percentage of occlusion, such as those where the screen is blocked by a fish, covering at least 70 % of its area. When this occurs, the area covering the screen is considered too large for a blob and four frames can be skipped.

```
1. [performance=processing_time, goal=classify video]: skip half of video
2. [screen blocked with big fish]: skip four frames
3. otherwise: go to next frame
```

6.7.8 Perform Video Classification

Finally, when classification is performed on the video, these values are computed and written on to the frame. In a video classification, the attributes that need to be computed are the brightness, clearness and green tone levels. However, for task that involves the combination of video classification and fish detection tasks, the presence (or absence) of fish in that video should also be indicated. Hence there are two variations of this method to cater for the presence of fish in a video.

```
1. [goal=classify video]: compute_write_average_brightness -->
   compute_write_average_clearness --> compute_write_presence_algae
2. otherwise: compute_write_average_brightness --> compute_write_average_clearness
   --> compute_write_presence_algae --> compute_write_presence_fish
```

The next section will illustrate some sample plans that have been generated using the paths outlined in this section, so as to give a general idea of the working of the planner and the complexity of the plans.

6.7.9 Plan Traces

Based on the paths available for different goals, detection algorithms and counting algorithms, among others, a few example plan traces are provided here to emphasise different paths generated by the planner. Each line in the trace includes a step number for readability, the tool that is invoked and its parameters. Firstly, recall the plan trace for video classification task given in Section 5.6.1:

```
01 grab_frame_image videos/2.mpeg 1
02 extract_RGB_colours Video_2.mpeg/1
03 compute_histogram Video_2.mpeg/1
04 compute_main_statistical_moments Video_2.mpeg/1/Hist_Array_1.dat
<-- Repeat steps 01-04 over frames -->
05 compute_write_average_luminosity Video_2.mpeg/
06 compute_write_average_clearness Video_2.mpeg/
```

```
07 compute_write_presence_algae Video_2.mpeg/
08 write_frames_to_output_video Final_2.mpeg.avi Video_2.mpeg/Output/
```

Plan Trace 1: Video classification according to brightness, clearness and algal levels.

The possible plan traces for this will not vary in terms of the processes involved, but rather in the number of frames being processed (iterations), *i.e.* the processes within lines 01-04 in the plan trace. In a video with 50 frames, for instance, this block will be performed over all the frames, *i.e.* 50 times, yielding a plan with 204 steps (50 frames x 4 iterative steps and 4 final steps). However, if the constraint criterion Performance is set to Processing Time, then only three frames are processed; the first, the middle and the last. In this case, only three iterations are required. The total number of steps generated for this plan is 16 (3 frames x 4 iterative steps and 4 final steps).

For the fish detection and counting task, several combination of algorithms can be generated for the final plan. Two examples are presented based on the different preconditions for selecting detection and counting algorithms, as well as preconditions for skipping frames based on presence of fish blocking screen.

```
00 preliminary_analysis videos/2.mpeg
01 grab_frame_image_videos/2.mpeg 1
02 extract_RGB_colours_Video_2.mpeg/1
03 compute_histogram_Video_2.mpeg/1
04 compute_main_statistical_moments Video_2.mpeg/1/Hist_Array_1.dat
05 create_gaussian_mixture_model videos/2.mpeg Video_2.mpeg/GMM/ 50
06 compute_connected_components_and_ratio_area Video_2.mpeg/1/CorrectBlobs.jpeg
   Video_2.mpeg/1/
07 compute_camshift Video_2.mpeg/1.jpg Video_2.mpeg/1/BlobsBW.jpeg Video_2.mpeg/1/
08 compute_write_num_fish_in_frame_and_video Video_2.mpeg/1/ Video_2.mpeg/1fish.dat
   Video_2.mpeg/1.jpg Video_2.mpeg/ 1.jpg
09 determine_fish_blocking_screen Video_2.mpeg/2/CorrectBlobs.jpeg Video_2.mpeg/1/
<-- Repeat steps 01-04 and 06-09 over the frames -->
10 write_frames_to_output_video Final_2.mpeg.avi Video_2.mpeg/Output/
```

Plan Trace 2: Fish detection and counting task using an adaptive background model (Adaptive Gaussian Mixture Model).

```
00 preliminary_analysis videos/2.mpeg
01 grab_frame_image_videos/2.mpeg 1
02 extract_RGB_colours_Video_2.mpeg/1
03 compute_histogram_Video_2.mpeg/1
04 compute_main_statistical_moments Video_2.mpeg/1/Hist_Array_1.dat
05 create_W4_model Video_2.mpeg/ Video_2.mpeg/W4/
06 extract_HSV Video_2.mpeg/1.jpg Video_2.mpeg/1/
```

```
07 compute_backprojection Video_2.mpeg/1/Hue.jpg Video_2.mpeg/1/
08 compute_connected_components_and_ratio_area Video_2.mpeg/1/CorrectBlobs.jpeg
   Video_2.mpeg/1/
09 compute_camshift Video_2.mpeg/1.jpg Video_2.mpeg/1/BlobsBW.jpeg Video_2.mpeg/1/
10 compute_closest_blob Video_2.mpeg/1/ Video_2.mpeg/1/
11 compute_write_num_fish_in_frame_and_video Video_2.mpeg/1/ Video_2.mpeg/1fish.dat
   Video_2.mpeg/1.jpg Video_2.mpeg/ 1.jpg
12 determine_fish_blocking_screen Video_2.mpeg/2/CorrectBlobs.jpeg Video_2.mpeg/1/
<--Repeat steps 01-12 over the frames-->
13 write_frames_to_output_video Final_2.mpeg.avi Video_2.mpeg/Output/
```

Plan Trace 3: Fish detection, counting and tracking task using a non adaptive background model (W4 model).

Plan traces 2 and 3 show different planning steps to perform two types of counting tasks. Plan Trace 2 has selected Gabor filter for computing texture features because the preferred algorithms were not available, used Adaptive Gaussian Mixture Model as its detection algorithm because it the best option with the given domain information, and selected the fish detection and counting algorithm without tracking based on the goal. As there were two instances where there was a fish blocking the screen, four frames were skipped each time (eight less frames). For a video with 50 frames, this plan contains 339 steps (Preliminary Analysis + Create and Apply Adaptive Gaussian Mixture Model + 42 iterations x 8 steps + Write Frames to Video). Plan Trace 3 has selected histogram and statistical moments calculation for texture features, used W4 model for detection because it has best matched the domain conditions, and also performed tracking and counting of fish in the video so far. Also, as there was no fish blocking the screen at any time, all the frames were taken into processing. For a video with 50 frames, this plan contains 602 steps (Preliminary Analysis + 50 frames x 12 steps + Write Frames to Video). In these examples, the algorithms selected and the number of iterations both vary, yielding different paths and number of planning steps.

6.8 Concluding Remarks

The main workings of the planner and workflow have been outlined in this chapter. The main contribution lies in the development of an enhanced HTN-based planner that utilises ontologies for tools selection. It can plan in automatic and semi-automatic (interactive) modes. In the semi-automatic mode, it presents all the available VIP tools that can perform a specific primitive task and provides recommendations for the user

to choose from, making it an informative and interactive tool. In this fashion, the user can be given some level of control during the planning phase. The mechanism adopts a planning interleaved with execution approach. A set of process models or HTN methods that contain the preconditions, decompositions and postconditions for non primitive tasks for three main VIP tasks were determined and encoded into the process library. The workflow enactor provides a higher level of abstraction by formulating the input to the planner and interacting with the user. The incorporation of the planner, process library and ontologies within the workflow system has made it possible for the various components to provide an intelligent mechanism for image processing-naive users to conduct automated assisted video analyses. The next chapter will present three experiments to evaluate the system's efficiency and adaptability, as well as user learnability using several typical video processing tasks.

Chapter 7

Evaluation

In Chapter 1, a set of claims were presented to address the research questions of this thesis. Chapter 3 outlined a hybrid approach incorporating ontologies and planning in a workflow composition and execution framework as a means to support the thesis claims. Chapters 4, 5 and 6 described the three major components of the framework in detail. This chapter evaluates the implemented system based on the integration of these key technologies, in the aspects of *efficiency, adaptability* and *user learnability*. A set of experiments were devised and conducted to provide evidence to support the thesis claims. First the evaluation criteria are laid out (Section 7.1), followed by the test data for the experiments (Section 7.2), the tasks and subjects identified for the experiments (Section 7.3), the set up and results of the experiments (Section 7.4). A rigorous analysis of the results are presented to conclude.

7.1 Evaluation Criteria for Overall System

The criteria for evaluation have been set based on the research questions and claims outlined in Chapter 1. The thesis claims are as follows:

- Automated support could be provided for image processing-naive users to perform VIP tasks in a *time-efficient* manner using a novel semantics- and planning-based workflow composition and execution framework. Using this approach is more efficient at solving VIP tasks than using traditional manual methods without loss of accuracy in the quality of the solutions.

- Conducting VIP tasks using multiple image processing executables in a workflow system is more *flexible* and *adaptable* towards users' changing needs than

constructing solutions using a single image processing executable as employed by conventional image processing systems.

- The planning- and ontology-based automated-assisted mechanism to compose and execute workflows for VIP tasks helps the user *manage* and *learn* the processes involved in constructing optimal solutions. This has not been possible within state-of-the-art workflow and knowledge-based vision efforts.

The evaluation criteria were formulated by taking into consideration factors such as diversity in user requirements, variety in the quality of the videos (lighting conditions, object movement, *etc.*) and vastness of the data made available. The following hypotheses were tested for correctness.

1. Automated support could be provided for users without image processing expertise to perform VIP tasks in a *time-efficient manner* using a novel semantics- and planning-based workflow composition and execution system without loss of accuracy in the quality of the solutions produced.

2. Constructing VIP solutions using multiple image processing executables employing planning and workflow technologies is more *flexible* and *adaptable* towards changing users' needs than modifying single executable programs.

3. The semantics and planning based automated-assisted mechanism to compose and execute workflows for video processing tasks helps the user *manage and learn* the processes involved in constructing optimal solutions.

As will be described in the next section, the hypotheses were tested using videos originating from an uncontrolled environment where the data is collected continuously, the quality of the videos vary considerably and the range of tasks and requirements are diverse, which provide a suitable test set for the evaluation of these hypotheses.

7.2 Data Set: Ecogrid Videos

The videos used for the evaluation of this thesis come from an ecological source in Taiwan. These videos are captured, stored and made accessible to marine biologists via the Ecogrid project [33] , coordinated by the National Center for High Performing Computing (NCHC), Taiwan. The project is a joint effort between NCHC and several local and international research institutes which provides a Grid-based infrastructure

for ecological research. Data is acquired using geographically distributed sensors in various protected sites such as Fu-Shan forest, Yuan-Yang lake, Ken-Ting national park and Nan-Jen-Shan site. The video streams collected have enabled analysis in underwater reef monitoring, among others. Interesting behaviours and characteristics of marine life such as fish and coral can be extracted from the videos by performing analysis such as classification, detection, counting and tracking. A sample consisting of 27 videos from 2004 are used for experimental purposes. A few image captures of the videos are presented in Fig. 7.1.

Figure 7.1: Sample shots from Ecogrid videos. From left to right: clear with fish, algal presence on camera, medium brightness with fish, completely dark and human activity.

As can be seen from the images, the videos were taken in an uncontrolled open sea environment where the degree of luminosity and water flow may vary depending upon the weather and the time of the day. The water may also have varying degrees of clearness and cleanness. In addition, the lighting conditions change very slowly, the camera and the background are fixed, images are degraded by a blocking effect due to the compression, and the fishes are regions whose colours are different from the background and are bigger than a minimal area (they are unusable otherwise). Furthermore, as algae grow rapidly in subtropical waters and on camera lens, it affects the quality of the videos taken. Consequently, different degrees of greenish videos are produced. In order to decrease the algae, frequent and manual cleaning of the lens is required. These videos represent a suitable test set for experimentation as the video descriptions, such as brightness, clearness, algal levels and movement of objects vary between the videos. The next section describes the tasks identified for this video set.

7.3 Tasks and Subjects

Generally, ecologists and marine biologists analyse videos manually, with the help of some computational tools, such as JWatcher [11], VirtualDub [58] and Observer [84], most of which are commercial. Among the tasks that they conduct include filtering out

videos that are unusable, *e.g.* videos that are too dark or too bright, followed by an observation of the number of marine life present in a certain time frame. In particular, for uncontrolled environments, the task of counting (and tracking) the number of existing fish in a video would be essential for further analysis such as fish population during a particular time of day or year.

Based on the video descriptions and other features displayed from the video captures, as well as discussions with marine biologists, several broad categories of tasks have been identified as useful for this test data; video classification, fish detection, counting and tracking. These are described below, in order of increasing complexity:

T0. Video display and basic properties capture (*e.g.* frame rate, creation date, frame images and candidate background images). This task is used mainly as a utility for displaying the results of other tasks.

T1. Video classification based on brightness, clearness and green tone (algal) levels.

T2. Fish detection and counting in individual frames.

T3. Fish detection, counting and tracking in video.

(a) T1: Video classification. (b) T2: Fish detection and counting in frames. (c) T1 & T3: Video classification, fish detection and tracking.

Figure 7.2: Sample visual results for video processing tasks.

The results are annotated to the original video in text and numerical format. For example, Fig. 7.2(a) shows the brightness, clearness and green tone values on the resulting video while Fig. 7.2(b) shows the detected fish and the number of fish in the current frame image on the top left (incorrect in this frame). These tasks can be conducted in a combined fashion for more sophisticated analysis, for instance T1 and T3 can be combined to conduct video classification, fish detection, counting and tracking. Fig. 7.2(c) shows an example result for this task. The final video is annotated

with brightness, clearness, green tone and fish presence values, as well as the number of fish in the current frame (number at the top left of image) and the number of fish so far in the video (number at the bottom left of image).

These tasks will be useful to extract higher level analyses such as the behaviour of fish at certain times of the day or year, suitable environmental conditions for fish presence, and maximum or minimum number of fish in a certain time frame. However, extracting useful characteristics and features from these videos would take too long to perform manually. One minute's clip requires approximately 15 minutes' human effort on average for basic processing tasks [19]. As will be shown in the following sections, the time taken for humans to conduct specific tasks manually will be compared against the time taken to conduct the same tasks with the automated-assisted workflow system to test efficiency. Experiments to evaluate system adaptability and user learnability are also demonstrated in the next section. The following subjects and systems were used for experimental purposes:

S1. An image processing-naive user who constructs the solution with the assistance of the workflow tool using full automatic and/or semi-automatic mode.

S2. An image processing-naive user who solves the goal manually.

S3. An image processing expert who constructs the solution using specialised tools (*e.g.* by writing an OpenCV program).

S4. A workflow modeller who constructs the solution using the workflow tool (*e.g.* by manipulating the process library and ontologies).

The next section describes the experiments conducted to evaluate efficiency, adaptability and learnability in accordance with the thesis hypotheses.

7.4 Experiments Description

To fulfill the evaluation criteria outlined in Section 7.1, experiments demonstrating performance gains of plan generation with the assistance of the workflow tool over manual processing and program generation from scratch were conducted. Performance gains are measured in CPU time, manual processing time, and quality of solutions (compared to those provided by marine biologists), where appropriate. The experiments were designed according to the principles outlined in [50], where first a hypothesis

and its counterpart null hypothesis are formulated, followed by the variables, conditions and subjects for the experiments, then the hypothesis is tested by providing measurement levels to report the results and finally an analysis to accept or reject the null hypothesis. Where appropriate, tasks and systems from Section 7.3 will be referred to. All the experiments were conducted on 50 frames of each video. For experiments to test efficiency (Section 7.5) and user learnability (Section 7.7), eight participants from a variety of backgrounds were selected as subjects. None of them possessed image processing expertise; three computer scientists and five non computer scientists. The backgrounds of the non computer scientists include medicine, history and archeology, physics, ecology and marine biology. The reason for having this variety was to test the usability and effects of the system on a mix of users, with technical expertise (computer scientists) and without, with domain expertise (ecologist and marine biologist) and without. These two experiments also required user-driven evaluation measures. A sample questionnaire devised for conducting both these experiments is provided in Appendix C. All experiments were conducted on a laptop operating on Ubuntu 8.04 with 512 MB RAM and 1.6 GHz processor speed.

7.5 Manual *vs.* Automatic Approaches for Video Classification Task

To demonstrate the claim that automated support could be provided for image processing-naive users to perform video and image processing tasks efficiently, the task completion time of the solution constructed using the automated-assisted tool would be faster than the task completion time of the solution constructed manually. This would make evident that the constructed workflow tool is able to produce video processing solutions for image processing-naive users in a time-efficient manner. Tests for efficiency and accuracy were performed to evaluate this claim.

7.5.1 Experiment Setup

In this experiment, subjects were asked to classify 10 videos according to brightness, clearness and green tone (algal) levels. First, they were required to conduct the task manually, and then using the automatic workflow tool. Using manual processing, each video was played using a video processing software where the subject may pause and repeat the video in order to determine the brightness, clearness and green tone levels.

The subjects record the classification value for brightness as “bright”, “medium”, or “dark”, the classification value for clearness as “clear”, “medium” or “blur” and the classification value for green tone level as “green” or “not green”. They were advised to select these classification values based on their own judgement.

Using the automatic tool, the workflow system was started up as outlined in Section 6.2.2 (Workflow Enactor Functions). Subjects selected the option to perform the task ‘Video classification according to brightness, clearness and green tone levels’. Then they were prompted to select between a fast and a slow processing mode. For each video, they were required to run the slow processing followed by the fast processing. They were given the freedom to perform this task automatically using just the fast processing mode when they were confident enough to do so, otherwise they would proceed to use the two modes on the videos. The reason for conducting the experiment in this fashion was to test the confidence of the subject in the quality of the fast processing. Further details will be provided in Section 7.5.5. The number of videos processed before the user switched to using the fast processing mode only was noted.

The fast processing algorithm was taken as the automatic processing time as it has been empirically shown to produce the same quality of results as the slow one (see Section 7.5.5). The CPU time taken to perform the task automatically and the time taken to perform the task manually were computed. The accuracy of the results were computed using the following procedure. As there were three classification criteria (brightness, clearness and green tone), each matching value of the automatic (system) or manual (subject) classification value with the ground truth was given a score of 1. For each video manipulated by each subject, a score of 3 would indicate 100% accuracy, in which case all three classification values (brightness, clearness and green tone) matched the recommended values. For all 10 videos manipulated by each subject, an average score was computed. A percentage was then calculated for the quality of the solution produced. The users were also asked three additional questions, whether they noticed any difference in the fast and slow automatic processing times, whether they found the automatic processing less tedious than the manual processing and which method would they prefer to use if the tasks were to be done frequently. Appendix C.1 contains the instructions given to the subjects, the additional questions for subjects to answer and a sample data set for this experiment. Eight participants that consisted of image processing-naive users were used for this experiment. A set of ground truth classification values were provided by a marine biologist as a baseline for comparisons with solutions produced by the experiment subjects.

7.5.2 Results

Subject	Automatic		Manual		Difference	
	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time d_e	Accuracy d_a
1	2.12	61.11	47.90	76.19	-45.78	-15.08
2	2.13	61.11	39.65	53.33	-37.52	7.78
3	2.09	61.11	40.12	25.00	-38.03	36.11
4	2.06	61.11	45.33	87.50	-43.28	-26.39
5	2.13	61.11	35.02	52.38	-32.89	8.73
6	2.14	61.11	48.25	80.00	-46.11	-18.89
7	2.06	61.11	37.20	66.67	-35.14	-5.56
8	2.02	61.11	17.95	52.78	-15.93	8.33
Average	2.09	61.11	38.93	61.73	-36.83	-0.62

Table 7.1: Time and accuracy of automatic versus manual processing, and their differences for video classification task.

Table 7.1 contains the average time measurements, accuracies of the results and the differences between automatic and manual processing for each subject who participated in the experiment. As explained in the previous section, the metrics were produced based on 10 videos processed by each subject, out of a total of eight subjects. There was a total of 27 videos, where each video was manipulated three times throughout the entire experiment. The classification result provided by a marine biologist was taken as the base line for the accuracy. Based on these values, statistical tests were performed to evaluate the efficiency of the methods of processing and the quality of the solutions produced by the methods.

7.5.3 Testing of Efficiency

Statistical hypothesis testing using the *t*-distribution was conducted to measure the dependencies between the results obtained for the times taken to conduct automatic and manual processing. The hypothesis, null hypothesis, independent and dependent variables for this test are given below.

- Hypothesis

  Image processing-naive users solve VIP tasks using the workflow tool faster than

performing them manually, *i.e.* automatic processing takes less time that manual processing.

- Null hypothesis
  There is no difference in the time taken for image processing-naive users to solve the task manually and with the workflow tool.

- Independent variables:
  – Data (27 Ecogrid videos of various quality).
  – Subjects and systems used for solving task:
    1) S1: Image processing-naive user performing task using automated tool.
    2) S2: Image processing-naive user performing task manually.
  – Type of subjects: 8 users without image processing expertise.
  – Task T1: Classify video according to brightness, clearness and algal levels.

- Dependent variables
  – Time taken to perform task manually versus time taken to perform task automatically.

Ground truth: Determined manually by marine biologist.

For this sample set, the two sample dependent t-test was performed to determine the t value and its corresponding p value in order to accept or reject the null hypothesis. A significance level of $p < 0.05$ was taken as an acceptable condition to reject the null hypothesis. The t value is given by Equation 7.1 below:

$$t = \frac{\bar{d_e}}{\sqrt{\frac{\sigma_{de}^2}{n}}} \tag{7.1}$$

where n is the sample size, $\bar{d_e}$ is the mean of the differences between the manual and automatic times and σ_{de} is the standard deviation of this mean. Based on the values of t and n, a significance level was computed. Taking the automatic and manual times collected for this experiment:

$$\bar{d_e} = -36.83 \qquad \sigma_{de} = 9.12 \qquad n = 8$$

$$t = \frac{36.86}{\sqrt{9.12^2/8}} = -11.43$$

The degree of freedom is set to $n-1$, which is 7. A value of $t(7)$ = -11.43 corresponds to a significance level of $p \ll 0.0001$ (according to the table for t). This means

that the null hypothesis can be rejected. It can be concluded that the efficiency of automatic processing is significantly higher than the efficiency of manual processing.

7.5.4 Testing of Accuracy

A similar statistical hypothesis testing using the *t*-distribution was conducted to measure the dependencies between the accuracies between the results obtained for automatic and manual processing. The hypothesis, null hypothesis, independent and dependent variables for this test are given below.

- Hypothesis
  The quality of the solutions produced by image processing-naive users when solving video classification task using the automatic workflow tool is higher than quality of the solutions produced when they perform the task manually.

- Null hypothesis
  There is no difference in the quality of the solutions produced by image processing-naive users to solve the task manually and with the workflow tool.

- Independent variables
  – Data (27 Ecogrid videos of various quality).
  – Subjects and systems used for solving task:
    1) S1: Image processing-naive user performing task automatically.
    2) S2: Image processing-naive user performing task manually.
  – Type of subjects: 8 users without image processing expertise.
  – Task T1: Classify video according to brightness, clearness and algal levels.

- Dependent variables
  – Quality of results as compared to ground truth.

Ground truth: Determined manually by marine biologist.

Again, the two sample dependent t-test was performed to determine the *t* value and its corresponding *p* value in order to accept or reject the null hypothesis. A significance level of $p < 0.05$ was taken as an acceptable condition to reject the null hypothesis. The *t* value is given by Equation 7.2 below:

$$t = \frac{\bar{d_a}}{\sqrt{\frac{\sigma_{da}^2}{n}}} \tag{7.2}$$

where n is the sample size, $\bar{d_a}$ is the mean of the differences between the accuracies of the manual and automatic solutions and σ_{da} is the standard deviation of this mean. Based on the values of t and n, a significance level was computed. Taking the automatic and manual accuracies collected for this experiment:

$$\bar{d_a} = -0.62 \qquad \sigma_{da} = 19.20 \qquad n = 8$$

$$t = \frac{-0.62}{\sqrt{19.20^2/8}} = -0.0913$$

The degree of freedom is set to $n-1$, which is 7. A value of $t(7)$ = -0.09133 corresponds to a significance level of p = 0.4649 (according to the table for t). This means that the null hypothesis cannot be rejected. Hence, the accuracy of manual processing, although slightly higher on average, is not significant enough to indicate that it is more accurate than the solutions produced by automatic processing.

7.5.5 Discussion

Before the system was tested on the subjects, several experiments were conducted using different numbers of video frames for automatic processing. The aim was to determine the minimum number of frames required to perform video classification where the quality of the result was the same as the processing that took into account all 50 frames. The less frames taken for processing, the less the CPU time taken to perform the classification task, the more efficient it becomes. As the brightness, clearness and green tone levels were computed based on values accumulated over the frames, the number of frames required for processing could be tuned so as to achieve an optimal threshold that could be used in the automatic system for user evaluation.

It was discovered that three frames of the video (represented by the first, middle and last frames) were sufficient to produce fast processing without loss of accuracy in the quality of the solution for all 27 videos (see Fig. 7.3). The graph also shows that the automatic processing times were not affected by the quality of the videos (brightness, clearness and green tone levels). One observation from the results is on the relationship between the manual processing times and accuracy. The accuracy of the subjects' manual classification varied slightly. It was noted that subjects who had domain knowledge (*e.g.* ecologist and marine biologist) had higher levels of accuracies in the video classification than their counterparts without domain expertise (*e.g.* computer scientists). However, they did not take less time in performing this classification. The regression

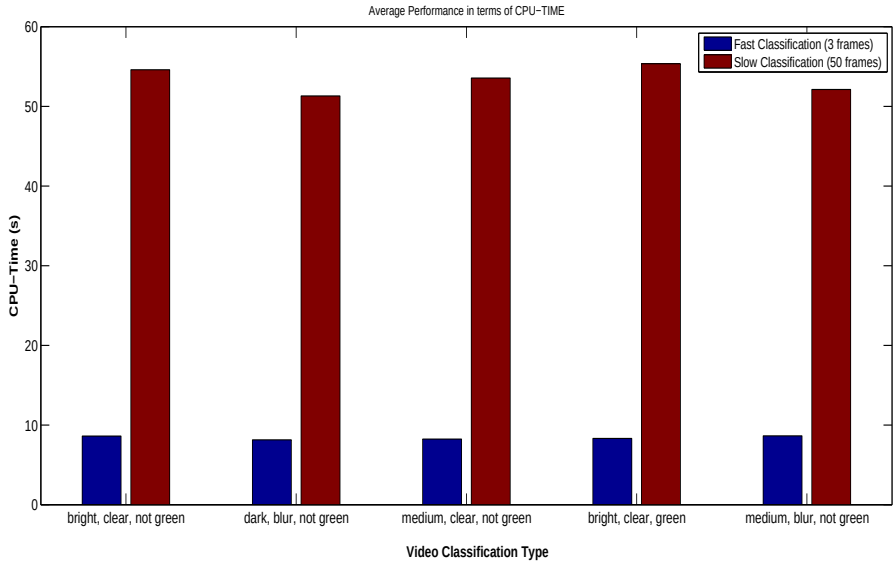


Figure 7.3: Average CPU-time for fast (using 3 frames) and slow (using 50 frames) processing for each type of video.

line in Fig. 7.4 suggests that more accurate classification was achieved when more time was spent performing the task.

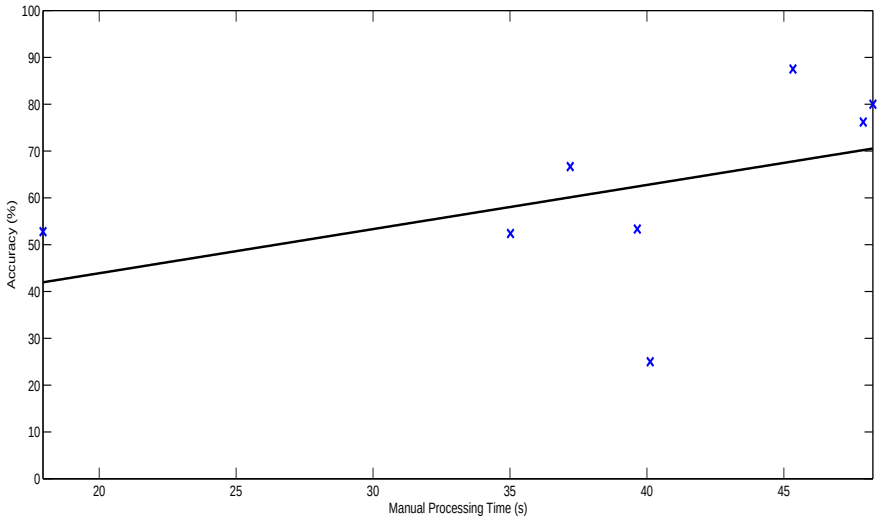


Figure 7.4: A scatter plot and least squares line regression of the times taken to perform video classification task manually and the corresponding accuracy levels produced.

The results obtained from the experiments to test efficiency and accuracy between manual and automatic approaches for video classification has been favourable for the

thesis claims. The outcome of the statistical tests and subjects' responses to the three additional questions in Appendix C.1 have led to the following findings:

- Automatic processing for video classification is on average 94.73% faster than manual processing without loss of accuracy in the solutions produced.
- 75% of the subjects found performing the video processing task using the automatic tool was less tedious than performing it manually.
- All subjects preferred to use the automatic tool for video classification over the manual method if they were to conduct the video classification task frequently.
- On average subjects learn to choose only the fast algorithm (which is as reliable as the slow one) after 6.4 runs in a sample of mixed types of videos. This will be discussed further in Section 7.7.4 (User Learnability).

The next experiment was aimed at testing the efficiency and accuracy levels in the workflow tool and conventional image processing approaches when it comes to adapting to changes in user preferences, with a focus on varying domain descriptions.

7.6 Single- *vs.* Multiple-Executable Approaches on Software Alteration

This experiment aims to show that the workflow system which adopts a multiple-executable approach adapts quicker to changes in user preferences than its single-executable counterpart (specialised image processing programs). This is the test of adaptability of the workflow system to reconstruct plans (solutions) efficiently when the user changes the domain descriptions for a task. This experiment will demonstrate that a solution constructed by an image processing expert using a single specialised image processing executable takes longer to produce given such changes in the domain descriptions for the same task.

7.6.1 Experiment Setup

In this experiment, an image processing expert and a workflow modeller have access to the same set of video and image processing tools; the former has an OpenCV program with available image processing algorithms written as functions and the latter in the

form of independent executables defined in the process library. These modules are the image processing components developed for this thesis, described in Section 5.5.

Both subjects were familiar with the systems that they were manipulating. They were given an identical task to perform – fish detection, counting and tracking in a video. Both systems were able to perform this task using a default detecting and tracking algorithm. In the workflow tool, the Gaussian mixture model was defined as the detection algorithm, no methods were defined for the selection of any other detection algorithm. In the OpenCV program, the Gaussian mixture model was used as the detection algorithm. Six scenarios were presented to both subjects containing changes to domain conditions (see Table 7.2). Both subjects were asked to make modifications or additions of code to their respective systems to cater for these changes in order to solve the VIP task as best as possible. For this purpose, they were both given which detection algorithm should be selected in each case. The number of lines of code (OpenCV for image processing expert and Prolog for workflow modeller) and the time taken to make these modifications were computed for both subjects. A line of code in OpenCV is represented by a valid C++ line of code, *i.e.* a line ending with a semi-colon (`;`), a looping or conditional statement (`if/for/while`). In Prolog, a line of code is a single predicate or fact ending with a full stop (`.`), a statement ending with a comma (`,`) or the head of a goal (line ending with `:-`).

The quality of the solutions is calculated as follows. There are two values to be considered, the first is the number of fish in the current frame and the second is the number of fish in the video so far. Each of these is given a score of 1 if there is a match with the ground truth. For each frame, the accuracy could be 0%, 50% or 100%. An average accuracy as a percentage is computed by taking the accuracy of 10 frames (1^{st}, 6^{th}, 11^{th}, ..., 46^{th}) from each video over all 27 videos.

7.6.2 Results

Table 7.2 contains the results obtained for this experiment. For each domain description altered, the time taken to modify the system, the number of lines of code added, and the accuracy of the solution are given.

The results produced are used to compare the efficiency of two different problem solving systems given equal starting points in the form of available solutions and equal expectations in the alterations required when user preferences change. Despite measuring the lines of code between two programming languages, the experiment does not

Domain Descriptions (User Preference)	Image Processing Expert			Workflow Modeller		
	New Lines of Code	Time (min.)	Accuracy %	New Lines of Code	Time (min.)	Accuracy %
Prefer false alarm than miss	43	16	58.25	3	3	59.30
Prefer miss than false alarm	56	23	62.55	2	2	64.80
Clear, no background movement	43	16	58.46	3	3	60.71
Clear, background movement	61	27	60.42	2	2	60.10
Blur, no background movement	43	16	60.88	3	3	62.09
Blur, background movement	57	32	63.80	2	2	61.22
Average	50.50	21.67	60.73	2.50	2.50	61.37

Table 7.2: Comparisons of number of new lines of code written, processing times and accuracies of solutions between single-executable image processing program and multiple-executable workflow system to adapt to changing domain descriptions.

intend to compare the two programming languages, but rather, to show the differences in effort required (*i.e.* time) to solve video processing problems using two different approaches (single- versus multiple-executable systems).

7.6.3 Testing of Efficiency

Statistical hypothesis testing using the t-distribution was conducted to measure the dependencies between the results obtained for the times taken to make changes to the workflow tool and OpenCV program. The hypothesis, null hypothesis, independent and dependent variables for this test are given below.

- Hypothesis
  Constructing VIP solutions using the workflow tool is faster than modifying existing image processing programs each time a domain description is altered for a fish detection, counting and tracking task, *i.e.* less time is taken to modify workflow tool than image processing program to adapt to changes in user preferences.

- Null hypothesis
  There is no difference in the time taken to solve the task using the workflow tool and modifying existing image processing programs each time a domain description is altered for fish detection, counting and tracking task.

- Independent variables
  – Data (27 Ecogrid videos of various quality)

– Subjects and systems used for solving task:

1) S3: An image processing expert solving the task by modifying an existing OpenCV program.

2) S4: A workflow modeller who solves the task by manipulating the ontologies and process library within the workflow system.

– Task:

T3: Detect, count and track fish in video with the following domain conditions:

1) Accuracy level prefer_miss_than_false_alarm.

2) Accuracy level prefer_false_alarm_than_miss.

3) Clear with no background movement.

4) Clear with background movement.

5) Blur with no background movement.

6) Blur with background movement.

- Dependent variables

  1) Time taken to modify existing OpenCV program versus time taken to encode changes in workflow tool.

  2) Number of new lines of code added to encode the changes in OpenCV program and workflow tool.

- Assumption:

  Image processing expert and workflow modeller have the same set of algorithms, the former in the form of OpenCV program with user-defined function calls, the latter as image processing executables represented in the process library.

Ground truth: Determined manually.

For this sample set, the two sample dependent t-test was performed to determine the t value and its corresponding p value in order to accept or reject the null hypothesis. A significance level of $p < 0.05$ was taken as an acceptable condition to reject the null hypothesis. Using the formula provided by Equation 7.1, where n is the sample size, $\bar{d_e}$ is the mean of the differences between the times of the workflow tool and image processing system, and σ_{de} is the standard deviation of this mean. Based on the values of t and n, a significance level was computed. Taking the automatic and manual times collected for this experiment:

$$\bar{d_e} = 19.17 \qquad \sigma_{de} = 6.69 \qquad n = 6$$

$$t = \frac{19.17}{\sqrt{6.69^2/6}} = 7.01$$

The degree of freedom is set to $n-1$, which is 5. A value of $t(5) = 7.01$ corresponds to a significance level of $p \ll 0.05$. This means that the null hypothesis can be rejected. Thus the workflow tool is faster to adapt to changes in domain descriptions than the image processing program.

7.6.4 Testing of Accuracy

Again, statistical hypothesis testing using the t-distribution was conducted to measure the dependencies between the results obtained for the accuracies of the solutions provided by the workflow tool and the OpenCV program. The hypothesis, null hypothesis, independent and dependent variables for this test are given below.

- Hypothesis
  Constructing VIP solutions using the workflow tool yields more accurate solutions than modifying existing image processing programs each time a domain description is altered for a VIP task.

- Null hypothesis
  There is no difference in the quality of the solutions obtained using the workflow tool and modifying existing image processing programs each time a domain description is altered for a VIP task.

- Independent variables
  – Data (27 Ecogrid videos of various quality)
  – Subjects and systems used for solving task:
  1) S3: An image processing expert solving the task by modifying an existing OpenCV program.
  2) S4: A workflow modeller who solves the task by manipulating the ontologies and process library within the workflow system.
  – Task:
  T3: Detect, count and track fish in video with the following domain conditions:
  1) Accuracy level prefer_miss_than_false_alarm.
  2) Accuracy level prefer_false_alarm_than_miss.
  3) Clear with no background movement.
  4) Clear with background movement.

5) Blur with no background movement.

6) Blur with background movement.

- Dependent variables

  1) Quality of solutions, assessed against manually determined values.

- Assumptions

  Image processing expert and workflow modeller have the same set of algorithms, the former in the form of OpenCV program with user-defined function calls, the latter as image processing executables represented in the process library.

Ground truth: Determined manually.

Again, the two sample dependent *t*-test was performed to determine the *t* value and its corresponding *p* value in order to accept or reject the null hypothesis. A significance level of $p < 0.05$ was taken as an acceptable condition to reject the null hypothesis. Using the formula provided by Equation 7.2, where *n* is the sample size, $\bar{d_a}$ is the mean of the differences between the quality of the solutions generated by the workflow system and the image processing system, and σ_{da} is the standard deviation of this mean. Based on the values of *t* and *n*, a significance level was computed. Taking the automatic and manual times collected for this experiment:

$$\bar{d_a} = 0.64 \qquad \sigma_{da} = 1.56 \qquad n = 6$$

$$t = \frac{0.64}{\sqrt{1.56^2/6}} = 1.01$$

The degree of freedom is set to $n-1$, which is 5. A value of $t(5) = 1.01$ corresponds to a significance level of $p = 0.1794$. This means that the null hypothesis cannot be rejected. Thus, the quality of the solutions produced by the workflow tool, although on average slightly better than the quality of the solutions of the image processing program, is not significant enough to be considered more superior.

7.6.5 Discussion

The graph contained in Fig. 7.5 shows the time comparison between the multiple-executable workflow tool and the single-executable image processing system when each domain description was altered. Clearly, from the graph and the results obtained from the statistical tests, the workflow system is much faster to adapt to these changes than the image processing system.

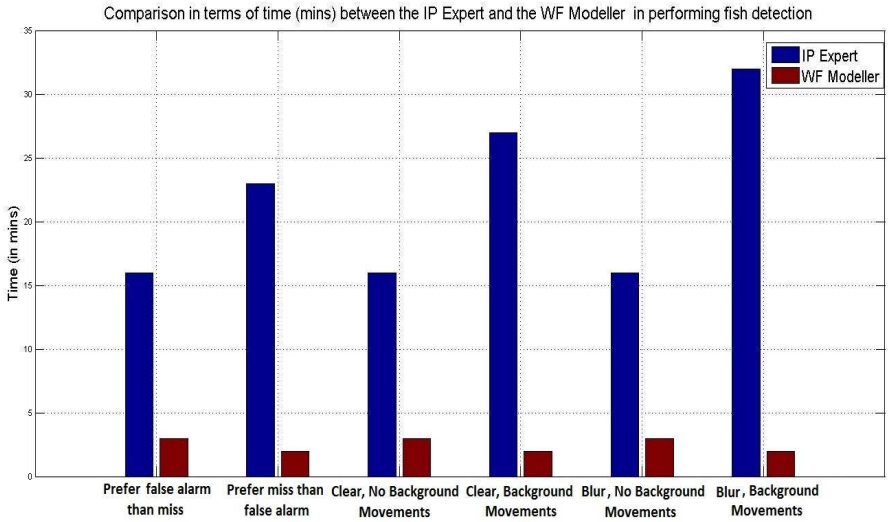
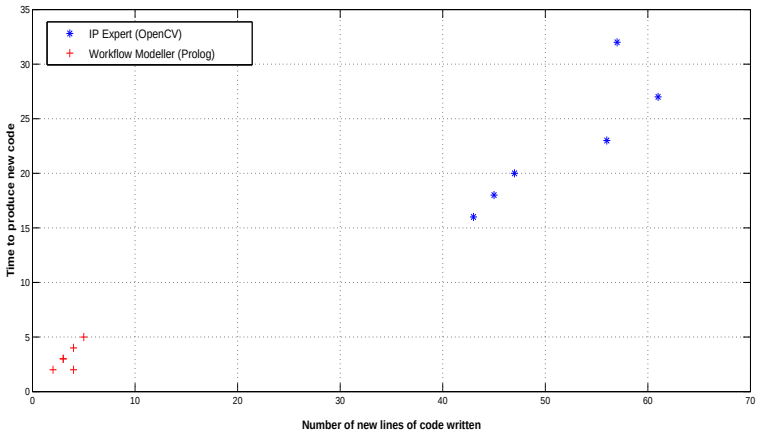


Figure 7.5: Plot of time taken by image processing expert and workflow system to adapt to changing domain descriptions.

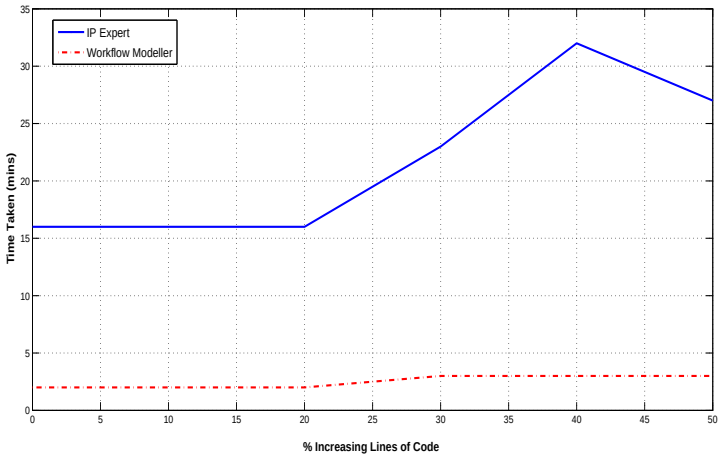
When an existing specialised image processing program can be found to support a specific task, the specialised program works very well. However, when the user requirements (domain descriptions) are altered this is no longer guaranteed without modifications to the program. This modification could range from 16 to 32 minutes for these tasks and initial OpenCV based program and take at most 61 lines of code to be written. The workflow tool, however, is adaptable to these changes very efficiently, taking just 3 minutes and 3 lines of code at most. The steps involved for the workflow modeller to encode these changes include adding a method in the process library to encapsulate the new domain descriptions as preconditions, and the relevant detection algorithm. Two more lines are added in the capability ontology to introduce this description as a performance criteria and to tie it to a relevant tool.

In order to perform the fish detection, counting and tracking task when domain conditions change, the image processing expert has to modify the OpenCV program to invoke an appropriate detection algorithm to cater for the changes. In particular, the most suitable background model creation algorithm has to be invoked. However, it involves more than just the invocation of the background model algorithm. The program also has to take appropriate actions in order to update the background model. As shown in the process models for creating and updating background models (contained in Figures 5.15 and 5.16), this mechanism would require conditional constructs (`if-then-else`). Besides, declarations of new variables, memory allocation and deallocation are also added for more complex data structures used, such as pointers. In the

workflow tool, such low level details do not need to be accounted for because they are already encoded in the independent executables and will be reused effectively.



(a) Number of new lines of code written *vs.* time taken for image processing program and workflow tool.



(b) Increase in lines of code written over time by image processing expert and workflow modeller.

Figure 7.6: Graphs to indicate the number and percentage of lines of code written for specialised image processing program and workflow tool for changing domain descriptions for fish detection and counting task.

Graph 7.6(a) shows the number of lines of code written and the time taken to produce them by the image processing expert and the workflow modeller. It can be seen that very few lines of Prolog code are added to the workflow tool and these changes are made quickly (indicated by the $+$'s at the low end of both axes). Whereas the

image processing program requires many more lines of OpenCV code and done over a much longer time (indicated by the scatter on the top right of the plot). Although OpenCV and Prolog have different features, as the former is based on procedural C++ and numerical while the latter is declarative and symbolic, the results of this experiment indicate that the declarative approach for solving VIP tasks using the multiple-executable workflow system is 86.46% more efficient than the procedural approach using single-executable image processing program.

Graph 7.6(b) indicates a slightly more interesting analysis beyond time comparisons. If the modifications required more effort, then the number of lines of code required for the image processing program is significantly higher than the number of lines of code required for the workflow tool. This increase is highlighted in the peak of the graph. This makes the workflow tool more efficient as a problem solver.

In terms of accuracy, the workflow tool on average performed slightly better than the image processing program, however, this difference was not significant enough to conclude that it produced solutions with better quality. Hence, without loss of accuracy, the multiple-executable workflow tool is a more adaptable problem solver than the single-executable image processing program.

7.7 User Learnability in Selection of Optimal Tool for Detection Task

In this experiment, the system's ability to help the user learn and manage the processes involved in constructing optimal video processing solutions is tested. An optimal tool is one that yields the best overall performance for the VIP task. If the workflow tool is run in full automatic mode, then it self-manages the creation of workflows for the user. This is achieved by making use of expert heuristics in assisting with the planning process. However, the system is not able to assess the optimality of the plan that results from this automatic solution as verifying video processing solutions computationally is not a trivial task. Humans, on the other hand, are able to assess the optimality of the plan by viewing the video containing the results. For example, it is trivial for a human to verify that the system has counted two fish in a frame by observing the count displayed in the resulting frame and the bounding boxes around the fish.

In order to allow the user to learn which tool options have worked the best, a semi-automated mode for workflow composition has been provided. This mode requires the

user to make a tool selection that is embedded in a workflow step when more than one option is available for performing a primitive task. Upon selecting a tool, the planning and workflow execution continues and the final result is displayed to the user. The aim of this experiment is to test whether the user can learn which is the best performing tool based on the descriptions provided by the system and the ability to compare the results of different tools for the same task. The set of instructions, data and questions posed to the user can be found in the questionnaire provided in Appendix C.2. Eight participants that consisted of image processing-naive users were used for this experiment, which included two domain experts – an ecologist and a marine biologist. These are the same subjects that were used in the experiment in Section 7.5.

7.7.1 Experiment Setup

In this experiment, each subject was presented with seven pairs of similar videos and seven pairs of dissimilar videos to perform a fish detection and counting task. Similar videos have the same video descriptions (brightness, clearness and green tone levels), while dissimilar videos have differing video descriptions. For these 14 pairs, it was determined previously that similar videos will use the same detection algorithms (hence same optimal detection tool) while dissimilar videos required different detection algorithms (hence not the same optimal detection tool) and have been used as baseline values for the evaluation. The subject was asked to perform this task using the *semi-automatic* mode of the workflow tool on all 14 pairs of videos. The ordering of the pairs were mixed between similar and dissimilar. The aim was to test if they were able to determine the most optimal tool for the detection algorithm based on the recommended descriptions provided by the workflow tool. If they were, then they should select this tool as the most optimal one in the next similar video presented to them. They should also not conclude to select this tool as the most optimal one in the next dissimilar video presented to them.

Before conducting the experiment, subjects were given a demonstration of one run using the semi-automatic mode to familiarise them with the procedures involved. Furthermore, they had performed the experiment in Section 7.5 and have some familiarity with the system. For each pair, they first conducted the task on the first video given. The workflow tool will display the video to the subject before proceeding to solve the task (see Fig. 7.7). Knowing the descriptions of the video (*e.g.* brightness, clearness, movement speed, presence of fish), the subject will have more information when se-

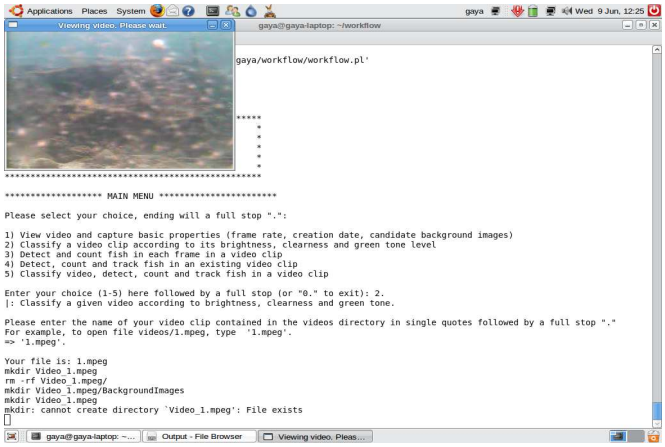


Figure 7.7: A video is displayed by the workflow tool to the user before conducting the task. This allows the user to capture its descriptions, if not known already.

lecting a detection algorithm with the help of the recommended descriptions provided by the workflow tool. After selecting a detection algorithm, the workflow tool will display the final result of this task based on this selection visually. The subject repeated the same task on the same video by trying another detection algorithm, if they wished, and observed the result.

When they were confident with the most optimal tool, they proceeded to perform the same task on the second video in the pair. This video could have similar video descriptions as the previous one (*i.e.* similar) or not (*i.e.* dissimilar). If the video descriptions are similar then the best tool inferred from the previous run should also be the best (most optimal) tool for the second run, otherwise it should not. During the second run, subjects were asked which tool they would choose (if any) based on what they have inferred from having conducted the experiment on the first video. Each time they selected the best tool inferred from the first video for the second video in the similar pairs, they were given a score of “correct”. Each time they selected the best tool from the first video for the second video in the dissimilar pairs, they were given a score of “incorrect”. If they were not able to infer the most optimal tool from the first run, and thus could not infer any tool for the second video based on the first, they were given a score of “incorrect”. The subjects were asked five additional questions on the usability of the system, contained in Appendix C.2. These included verification of the system’s design principles, which included the option to provide constraints and video descriptions, usefulness of the recommended descriptions, and ease of use.

7.7.2 Testing of User Learnability

Statistical hypothesis testing using the *t*-distribution was conducted to measure the dependencies between the results obtained for the number of times the user selected the correct tool based on a previous similar video and the number of times the user selected the incorrect tool based on a previous dissimilar video. The hypothesis, null hypothesis, independent and dependent variables for this test are given below.

- Hypothesis
  The semi-automated mechanism to compose workflows for fish detection and counting task helps the user manage and learn the processes involved in constructing optimal solutions, *i.e.* users can learn the most optimal detection algorithm based on the descriptions provided by the workflow tool.

- Null hypothesis
  The descriptions provided by the workflow tool using the semi-automated mechanism to compose workflows for fish detection and counting task do not assist the user to learn the most optimal detection algorithm.

- Independent variables
  – Data: 14 pairs of videos from 27 Ecogrid videos, 7 similar pairs and 7 dissimilar pairs.
  – Subjects and systems used for solving task:
  1) S1: IP naive user who performs the task using the workflow tool.
  – Type of subjects: 8 users without image processing expertise.
  – Task:
  T2: Detect and count fish in frames.

- Dependent variables
  – Number of times correct tool was selected as optimal tool in second video for similar pairs of videos.
  – Number of times incorrect tool was selected as optimal tool in second video for dissimilar pairs of videos.

7.7.3 Results

Table 7.3 shows the results obtained to assess the level of user learnability when using the workflow tool in seven pairs of similar videos and seven pairs of dissimilar videos. A "correct" choice was made when the subject selected the correct optimal tool in the second of the pair of similar videos, while an "incorrect" choice was made when the subject selected the optimal tool inferred from the first video as the optimal tool in the second of the pair of dissimilar videos.

Subject	Similar Pairs No. correct choices c	Dissimilar Pairs No. incorrect choices i	Difference $c-i$ d
1	6	2	4
2	4	3	1
3	5	2	3
4	6	1	5
5	4	3	1
6	6	2	4
7	4	2	2
8	5	3	2
Average	5	2.25	2.75

Table 7.3: Number of times "correct" tool was selected in seven similar pairs of videos, number of times "incorrect" tool was selected in seven dissimilar pairs of videos.

For this sample set, the two sample dependent t-test was performed to determine the t value and its corresponding p value in order to accept or reject the null hypothesis. A significance level of $p < 0.05$ was taken as an acceptable condition to reject the null hypothesis. The t value is given by Equation 7.3 below:

$$t = \frac{\bar{d}}{\sqrt{\frac{\sigma_d^2}{n}}} \tag{7.3}$$

where n is the sample size, $\bar{d}$ is the mean of the differences between the number of correct times an optimal tools was selected in similar pairs and incorrect times taken in a sample dissimilar pairs and σ_d is the standard deviation of this mean. Based on the values of t and n, a significance level was computed. Taking the automatic and manual times collected for this experiment:

$$\bar{d} = 2.75 \qquad \sigma_d = 1.39 \qquad n = 8$$

$$t = \frac{2.75}{\sqrt{1.39^2/8}} = 5.59$$

The degree of freedom is set to $n-1$, which is 7. A value of $t(7) = 5.59$ corresponds to a significance level of $p = 0.0004$. This means that the null hypothesis can be rejected. With this, it can be concluded that subjects, more often than not, selected the correct (optimal) tool when they were presented with a similar video and did not choose this tool by chance. Hence, the semi-automatic mode has helped the user learn and manage the processes involved in selecting the optimal steps when solving a task.

7.7.4 Analysis

The results in Table 7.3 indicate that subjects were able to select the correct optimal tool when they were presented with a similar video 5 out of 7 times on average (71.43% of the time). This is relatively high compared to the instances when they incorrectly chose an optimal tool learnt from a video for another video that is not similar to the one where they have inferred the tool from, which was 2.25 out of 7 times on average (32.14% of the time). The statistical test proves that the workflow tool is indeed able to help subjects learn optimal VIP tools without having any image processing knowledge, but purely from their own visual capabilities in judging visual descriptions and from the textual descriptions provided by the system.

The user's understanding is tested via the provision of the description of the tools when there was more than one tool to perform a task. By repeating the same task using a different tool, the user can gain insight into how a different solution can be produced and could then 'judge' which tool should be used for producing the most optimal solution. Hence when a new video is presented and the same task is performed, the user would have learnt and gained confidence in selecting the most suitable tool. The user can evaluate the performance of any particular tool that they have selected for a particular task. Using this feedback loop, the system can now learn which tools work better for which type of videos (according to their video descriptions) and under which constraint conditions.

To further validate the system's usability and learnability features, the findings of the questionnaire are summarised as follows:

- All subjects agreed on the appropriateness of having the option to provide constraints and video descriptions to specify the VIP task in more detail.

- All subjects found the recommended domain descriptions suitable and helpful in assisting them make more informed decisions when selecting the best detection algorithms (*i.e.* VIP tools).
- All subjects felt it was appropriate to be given the control to select tools despite not having image processing expertise.
- On average, subjects were able to make the decision to run the fast processing option for video classification task after just 6.4 runs.
- All subjects would prefer to use the automated tool if they were to perform video classification and fish detection and counting tasks frequently.

These findings indicate that although the workflow tool and the domain that it manipulates are both complex, users without expertise in workflow or image processing domains can learn how to use the workflow tool and they can even learn some aspects of solving image processing problems via selection of VIP tools. These were achieved in a short period of time as indicated by the experiment's results. The experimental findings also verify and validate the strength of the integrated approach. The efficiency and accuracy of the results obtained indicate that the workflow system is able to produce results comparable to those produced by humans and by competing systems (*e.g.* image processing programs). The use of ontologies has been proven to be beneficial via the provision of recommended descriptions that were useful to users. The planner's correctness and completeness has been tested on a set of VIP tasks (goals), constraints and detection algorithms.

7.8 Summary

The evaluation of the integrated workflow composition and execution system has led to three major findings: The workflow system is 1) more efficient than manual processing without loss in accuracy, 2) more adaptable than conventional image processing programs when domain descriptions are altered and 3) enables image processing-naive users learn the best tools involved in producing optimal solutions. In addition, user-driven evaluations proved to be valuable to verify the system's usability and learnability. In particular, image processing-naive users strongly preferred to have automated assistance for conducting video classification and fish detection tasks should the tasks be done on a frequent basis. The level of accuracy of full automatic processing can

be verified via the use of the semi-automatic options first. Once the user has gained confidence in the system's capability to choose optimal solutions, the full automatic option could be used. Marine biologists have verified that suitable characteristics for such an integrated tool include reliability of results, repeatability (having the option to try again with different parameter values) and the possibility of running processes in batch mode. While the first two characteristics are already exhibited by the system, it is not able to run a task on a set of videos (batch mode). While this would be a straightforward engineering effort, it could also suggest a step towards providing parallel processing.

The core part of the framework has allowed a set of videos originating from an ecological source to be tested on a range of typical video processing tasks. Furthermore, the algorithms developed for the framework have been tested by image processing experts for the detection of humans and vehicles with minor extensions. This shows that the framework could be extended for different applications, potentially providing impact to a broader range of problem domains.

Chapter 8

Conclusions

This thesis has demonstrated how complex video processing problems can be performed efficiently with automated support for users who do not possess image processing expertise. A novel workflow composition and execution framework that heavily relies on two key technologies, ontologies and planning, was devised, implemented and evaluated. The hybrid framework was first introduced in Chapter 3, followed by the ontologies (Chapter 4), image processing components (Chapter 5) and finally the planner enhanced with workflow execution and interactive capabilities (Chapter 6). Experiments to evaluate efficiency, adaptability and user learnability measures were presented in Chapter 7. The integration has resulted in a semantics-rich and flexible mechanism for conducting video processing tasks, in particular for data that vary in quality and for user requirements that change. This is beneficial for the scientific, research and application communities. This chapter concludes by highlighting the main accomplishments of this thesis, followed by a summary of its strengths. These will then lead to future directions that this work could take from here.

8.1 Main Contributions

Being a cross disciplinary field of research, this work has entailed many interesting and vital aspects throughout the duration of its course. These include visiting NCHC Taiwan for gathering data and acquiring advice from marine biologists, collaborating with image processing experts on reviewing several hundred modules within approximately a thousand lines of OpenCV code, transforming them into 30 independent executables which could then be exploited in a workflow composition and execution context. The thesis started out by posing several research questions that needed to be addressed:

1. What are the requirements for a suitable system that would provide automated assistance for image processing-naive users to conduct a range of typical video processing tasks that would be too time-consuming to do otherwise?

2. Ontologies are generally useful for representation and inference in many applications today. What is a suitable form of ontology to represent the relevant concepts and relationships in the video processing problem domain?

3. Planning technologies have been successful at solving complex problems with well-defined goals using action sequences. What type of planning approach would be suitable to be used (domain-independent *vs.* domain-dependent, classical *vs.* decomposition-based planning) for generating solutions for VIP tasks?

4. How is a framework implemented in combination with a planner and ontologies for workflow composition and execution of VIP tasks?

Consistent with the set of claims outlined in Chapter 1 and validated in Chapter 7, these questions have now been answered. The key contributions of this thesis are presented in the next four subsections.

8.1.1 Novel Mechanism for Automatic Workflow Composition

The most important accomplishment of this thesis is in its contribution towards the growing field of automatic workflow composition. As highlighted in Chapter 1, much work is still needed in this area, especially in the e-Science context where more and more workflows are manipulated by users not technically adept in the domain that they are working on. The novel workflow composition mechanism proposed by this thesis is part of a hybrid three-layered workflow composition and execution framework that utilises two fundamental technologies, **ontologies** and **planning**. The key innovation that enables the automatic composition of workflows is integration.

8.1.2 Higher Efficiency in Automated-Supported Video Processing

Another major impact of the work of this thesis is on the provision of a more time-efficient method for performing video processing tasks, in particular for image processing-naive users. Traditionally they conducted these tasks manually which was extremely time-consuming, tedious and repetitive. With the provision of automatic support, a speed up of over 90% has been achieved without loss of accuracy in the solution.

8.1.3 New, More Flexible Approach for Video Processing

This thesis has introduced a new way of solving video processing problems by the use of multiple image processing executables. Conventionally, video processing problems are solved automatically with the aid of computer vision programs that are tailored to work on a small subset of videos with a high rate of accuracy in the solutions. These programs are written as single executables which are compiled and executed on videos. As has been made evident in Section 7.6, the multiple-executable system proposed by this thesis is more **flexible** and **adaptable** to user preferences than the single-executable system without loss of accuracy in the solution.

8.1.4 Construction of Optimal VIP Solutions by IP-naive Users

A significant impact of this work is on the image processing-naive community who can now **learn** how to solve video processing problems. Previously, video processing problems could only be solved computationally by those who had image processing expertise using highly specialised tools. The automated assisted framework developed for this thesis has enabled other users to have access to these tools as well via a planner that can operate in semi-automatic mode. Futhermore, users can now make informed decisions on which tools to select at a particular time point with the aid of recommended domain descriptions provided by the ontology that has been built for this thesis. As has been made evident in Section 7.7, users are able to learn which tool selections can lead to better results in solving video processing problems, which would have been impossible with any state-of-the-art systems prior to this.

8.1.5 Enhanced HTN-Based Planner

Another contribution is the implementation of a novel **HTN planner enhanced with execution capabilities**. This planner is domain independent and has been tested on the video processing domain. Among its key features include interleaving planning with execution, loop handling and has the ability to plan automatically or semi-automatically (interactive). The semi-automatic mode involves intervention from user for tool selection. This interaction is crucial as it gives user the **control** to make decisions and choices of actions based on their own judgements. This adds to the growing body of scientific research within the planning community.

8.2 Strengths and Limitations

The main strengths of the workflow composition and execution system lie in its **efficiency** (Section 7.5), **flexibility** and **adaptability** (Section 7.7) and **user learnability** (Section 7.7) in conducting video processing tasks as compared to conventional practices. This has laid the foundation for automatic video processing for users without image processing expertise. The power of **integration** has enabled several key techniques to be utilised to maximal effect within a hybrid framework. The components are loosely coupled with one another and can be easily extended to cater for new functions and elements, such as the addition of new concepts, relations, rules, goals and tools.

While the approach has several prominent features, it also has its limitations. As the work has focused on constructing a suitable ontology and a planner, and modelling the video processing domain as a planning problem, the system has not been implemented in a **distributed** context, e.g. in a web or Grid services context. However, the framework is compatible to be used within a distributed environment [77]. This could be easily seen with the notion of ontologies for consistent and shared vocabularies. In a distributed context, resource allocation for the operators will need to be catered for. The system also follows a sequential mode of processing where only one tool or operation can be performed at a time. Processes that do not depend on one another could be parallelised, appropriate with distributed processing. This would save computational speed, making the workflow even more efficient. With an external **parallel** computing facility this capability could be added.

A challenge faced by this work is that significant effort was required in domain modelling for video processing applications. The suitable domain descriptions were formulated with advice from marine biologists and image processing experts while the HTN methods were derived based on image processing experts' heuristics. This is clearly a tedious task for the workflow modeller and perhaps the efforts required could be reduced by automated means. The next section will probe some possible directions for this research to explore from this point onwards.

8.3 Future Directions and Reflection

Based on the achievements, strengths and limitations, several directions that this research may take are provided here. One clear direction is to extend this workflow composition and execution framework onto a distributed infrastructure, *e.g.* the Grid.

This would enable distributed processing, especially where different resources could be utilised for problem solving. One such capability has been exhibited by Caton *et al.* [18]. Existing major Grid workflow systems have the ability to do so. The approach undertaken by Pegasus, for instance, could be adopted for this purpose. The grid infrastructure could be provided by Taiwan's NCHC via the Ecogrid project. Most of the work is anticipated to involve engineering work which would involve workflow resource discovery and allocation. The possibility to merge with existing Grid workflow systems such as Triana, Taverna and Kepler should also be considered.

Another challenge to be addressed is to reduce the domain modelling efforts for video processing. The HTN methods used in the process library were derived from image processing experts' heuristics. However, these methods could also be derived automatically during a training phase. Using user's evaluation measures, the domain descriptions for constructing optimal solutions (those with the most accurate video processing results) could be *learnt*. The domain descriptions that match the derivation of optimal solutions could be encoded as preconditions for the methods, as opposed to relying on experts' heuristics alone. Some preliminary work on this machine learning approach has been undertaken by the HTN planning community [47, 59, 119], which could provide foundation for this research to take lead from.

The ontology constructed for this thesis has taken into account the goal, video descriptions and capabilities (tools) aspects. During the construction phase, the difficulty of entangling the application specific concepts with those that can be generalised was identified. The granularity and complexity level decisions of the ontology were key factors in the ultimate aim of developing an efficient planner. As the ontology evolves, *e.g.* with the incorporation of new concepts and relations, its complexity will become an issue, especially its use in conjunction with the planner. Hence, a robust taxonomic structure and a suitable granularity level that would allow the planner to search for useful outcomes without having to explore a very large set of alternatives should continue to be priorities. A possibility for the ontology to be extended and enriched would be by incorporating features and tools from existing ontological efforts, such as the Marine Metadata Interoperability Project [72] and the NeOn project [81].

An interesting question that arises is would the different practices adopted by different image processing experts be an inhibiting factor for the overall sucess of this methodology. While different motion analysis experts could opt to use different types of algorithms for certain parts of the processing, *e.g.* detection or tracking algorithms, the overall method of solving the problem would not vary by much, that is to say, that a

majority of motion analysis would include a pre-processing step, followed by the processing of each frame of the video and finally an aggregation of results accumulated over the frames. Should there be a new overall way of solving the same task that does not include these principal steps, then it could be introduced as an alternative in the process library and the results of the two different approaches could be compared.

Albeit at present the system is able to perform video classification, fish detection and counting tasks, it would be appropriate to extend it to include fish classification tasks. Several species of fish[1] have been identified in the Ecogrid videos and the provision of this facility would be an important step towards understanding the behaviours of different species. This extension is possible with the introduction of new executables customised for specific fish classification tasks, as well as the reuse of existing ones for the remaining video processing subtasks.

Optimistically, this framework could eventually be used for general-purpose video processing, not just confined to underwater videos. It could also serve as an educational tool for naive users on a larger scale. As with most interdisciplinary work, continuous collaboration with image processing experts and marine biologists will be vital for the overall success of this research. All the avenues for extensions of this work will be explored in the Fish4Knowledge project starting in October 2010.

[1] Sample images can be accessed at: `http://homepages.inf.ed.ac.uk/s0450937/ecoImages`.

Appendix A

Tables of Input, Output and Related Ontologies

A.1 View Video

Input	Output	Comments
video filename (string)	displayed video	Displays first 50 frames of video.
	frame rate	Stored in 'Video_X/view_video.dat'*
	date (string)	Stored in 'Video_X/view_video.dat'*
	bg images (jpeg)	10 background images stored in 'Video_X/BackgroundImages/'*
	images (jpeg)	First 50 frames of video are stored in directory 'Video_X/Images/'*

*Where X comes from the video filename, X.mpeg.

Notes:

1. Conservative filtering is applied to remove noise to the images before any processing is done, *e.g.* capturing background images.
2. This module is able to detect completely dark videos, in which case further system processing will stop.
3. The possibility of catering for high definition videos, *e.g.* 16-bit pixel frames should be investigated.

A.2 Preliminary Analysis

Input	Output	Comments	Ontology
video filename	initial brightness level	low/medium/high	domain
(string)	initial clearness level	low/medium/high	domain
	initial green tone level	low/medium/high	domain
	initial speed of movement	low/medium/high	domain
	texture feature: variance	double	domain
	texture feature: skewness	double	domain
	texture feature: uniformity	double	domain
	texture feature: entropy	double	domain
	percentage background object	low/medium/high	domain
	background movement	low/medium/high	domain

All output values are stored in X.dat where X is taken from video filename, X.mpeg.

A.3 Grab Frame Image

Input	Output	Comments
video filename	current frame image (jpeg)	
frame number		

Resulting grabbed frame image is stored in Video_X, where X is taken from the video filename X.mpeg. The image for the currect frame is required for all other processing.

A.4 Extract RGB Colours

Input	Output	Comments
current frame image	colour values	red, green, blue and yellow channels
	(4 images)	

The resulting images are saved as red.jpeg, green.jpeg, blue,jpeg and yellow.jpeg in the directory 'X/' where X is taken from the current frame image, X.jpeg.

A.5 Compute Histogram

Input	Output	Comments
current frame image	histogram value (array of int)	
show hist (yes/no)*	histogram representation (image)	User may select to view histogram

*This value is set to "yes" by default and **not** catered for in OpenCV nor the process library.

A.6 Compute Main Statistical Moments

Input	Output	Comments
histogram value* (array of int)	mean (string) variance (string) third moment (string) fourth moment (string) entropy (string) uniformity (string) clearness (string)	

*The input values are contained in a textfile produced by Compute Histogram (Section A.5). The output values are stored in a textfile with the same name as the input file in the present directory.

A.7 Compute Gabor Filter

Input	Output	Comments
current frame image scale* (int) angle* (0-360)	Gabor filter displayed mean variance	For each scale and angle one complex image will be computed and from this the mean and variance are extracted. Loop executed 16 times per frame (scale: 1, 3, 5, 7 and angle: 0, 45, 90, 135) to extract a vector of complex images.

*Scale is set to 3 and angle to 45 by default. However, values could be explicitly provided during execution.

A.8 Create Gaussian Background Model

Input	Output	Comments	Ontology
directory with background images* (string)	(2 images)		

*Typically Video_X.mpeg/BackgroundImages/ where X is the name of the video.

A.9 Create Gaussian Mixture Model

Input	Output	Comments	Ontology
current frame image directory (string)	background model (2 images)		capability

A.10 Create Moving Average Model

Input	Output	Comments	Ontology
current frame image directory learning speed (alpha)*	background model (1 image)		capability

*Typically this value is dependent on the inital speed of movement. This value is set to 0.020 for now.

A.11 Create Intra-Frame Difference Model

Input	Output	Comments	Ontology
input directory** output directory	background model* (2 matrices)		capability

*Three images have been created; mean image, distance image and intra frame difference.

**Input directory contains ALL the frames of a video.

A.12 Create W4 Model

Input	Output	Comments	Ontology
input directory** output directory	background model* (3 matrices)		capability

*Four images have been created; mean image, distance image, n image and W4 (RGB) image.

**Input directory contains background frames of a video.

A.13 Create Poisson Model

Input	Output	Comments	Ontology
input directory* output directory	background model** histogram file	histogram.dat in Output directory	capability

*Input directory contains all the frames, typically Video_X.mpeg/

**Output directory contains 4 images; lambda, lambda L, lambda R and Poisson frame, typically Video_X.mpeg/Y/Poisson/

A.14 Create Adaptive Poisson Model

Input	Output	Comments	Ontology
video* root video directory	background model**		capability

**E.g.* videos/1.mpeg

**Poisson images are contained in Video_X.mpeg/AdaptivePoisson/

A.15 Update Moving Average Background Model

Input	Output	Comments
current frame image old background model (image) learning rate (alpha)*	updated background model (jpeg)	From A.10 above

A.16 Fuse Background Images

Input	Output	Comments
frame image1 frame image2	Fused image	

A.17 Detect Moving Objects

Input	Output	Comments
background model type* current frame image background model image directory to store output image	image with identified blobs** (black and white)	 1, 2 or 3 images Same as where background image is

*1: W4, 2: Intra Frame, 3: Poisson, 4: Gaussian.

**Saved as Binary_Image.jpeg in the directory specified by the 4th input.

A.18 Perform Morphological Operation

Input	Output	Comments
image with identified blobs (from A.17) directory for output	image with potential correct blobs* (used as input for A.21)	Applicable only for some background models

*Output is stored as CorrectBlob.jpeg in output directory (*e.g.* Video_1.mpeg/2/).

A.19 Extract HSV values

Input	Output	Comments
current frame image output directory*	3 images for hue, saturation and value channels	Called Hue.jpeg, Saturation.jpeg and Hsv.jpeg respectively

*Directory is in the form Video_X.mpeg/Y/ where X is video name and Y is current frame number.

A.20 Compute Backprojection

Input	Intermediate Output	Output	Comments
hue channel image (from A.19) directory to save result*	histogram of hue channel (array of int stored in a textfile)	backprojection of hue plane (image)	

* In the form Video_X.mpeg/Y/. File(s) saved are Hue_backprojection.jpeg and Hist_Array.dat.

A.21 Compute Connected Components and Ratio Convex Hull Blob

Input	Output	Comments
image with potential correct blobs (A.18) directory	image with blobs* number of blobs** ratio area convex hull over blob**	blobs are different-coloured

*Two images produced in directory, BlobsBW.jpeg and BlobsColour.jpeg.

**Contained in number.dat.

A.22 Compute Camshift

Input	Output	Comments
current frame image image with blobs (from A.21) output_directory	centre (x,y)* orientation* size (width, height)* 	 Typically Video_X.mpeg/

*Stored in output_directory/blob_camshiftZ.dat where Z is the blob number.

A.23 Compute Closest Blob

Input	Output	Comments
frame directory 1 frame directory 2	blob pairs with minimum Euclidian distance*	

*The matching "closest" blob pairs between the two frames are appended to output.dat

Note: Typically a segment has 10 frames. If the present frame number is 10 or less, then the number in the segment is one less than it.

A.24 Compute and Write Number of Fish in Frame and Video

Input	Output	Comments
frame directory file with blob number and ratio frame image	frame image with text* total fish in video so far**	Video_X.mpeg/Y/ Video_X.mpeg/Y/number.dat Video_X.mpeg/Y.jpeg

*Saved as FishFrameText.jpeg in directory specified by first input (frame directory).

**FishesNumberTotal.dat (total fish count in video) is saved in frame directory.

A.25 Determine Presence of Fish Blocking Screen

Input	Output	Comments
image with correct blobs (*e.g.* from A.21) frame_directory	(0/1)*	If true, 4 frames will be skipped.

*Stored in frame_directory/block.dat

A.26 Compute and Write Average Luminosity

Input	Output	Comments
video directory*	final frame image	“Bright”, “Medium” or “Dark” written onto final frame.

*Typically in form Video_1.mpeg/ where each jpeg file in this directory will be considered by the executable. Their respective directories will be accessed for the statistical moments file to compute the average luminosity value which will be stored as the image BrightnessText.jpeg.

A.27 Compute and Write Average Clearness

Input	Output	Comments
video directory*	final frame image	“Clear” or “Blur” written onto final frame.

*Works using the same principle as A.26.

A.28 Compute and Write Presence of Fish

Input	Output	Comments
video directory*	final frame image	“Fishes” or “No Fishes” written onto final frame.

*Works using the same principle as A.26.

A.29 Compute and Write Presence of Algae

Input	Output	Comments
video directory*	final frame image	“Green” or “Not Green” written onto final frame.

*Works using the same principle as A.26.

A.30 Write Frames to Video

Input	Output	Comments
final video name* directory with frame images**	final video	

*Should be ended with extension .avi

**All frames beginning from 1.jpg will be taken to create new video.

Notes:

- All videos are stored in the directory *videos/*.
- All VIP tools are stored in the directory *tools/*.
- Directories with Video_<filename> should be full filenames, i.e. with the mpeg extensions, *e.g. Video_1.mpeg/*.
- Frame directories are in the form Video_<Filename>/<FrameNo>/ *e.g.* the directory for frame number 3 of the video 1.mpeg is *Video_1.mpeg/3/*.

Appendix B

Process Model Diagram

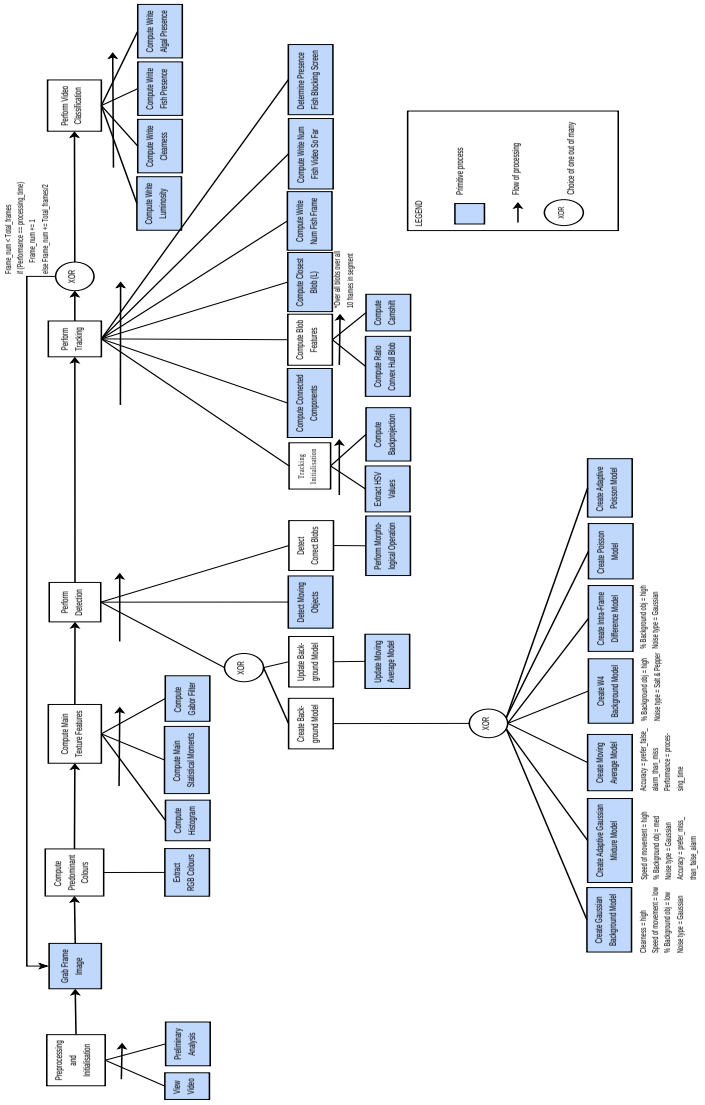


Figure B.1: Process Model for Video Classification, Fish Detection and Counting Task.

Appendix C

Evaluation Sheet

This is a questionnaire provided to the user to test the hypotheses for efficiency and learnability.

Type of User: Marine Biologist/Ecologist/Image Processing Expert/Other
Experience with System: Novice/Some Experience/Expert

In this study you will be asked to perform several tasks related to analysing 10 video clips. You will be asked to perform these tasks with the assistance of an automatic tool, and also manually. The tasks that you will conduct include the following below. More detailed instructions will be provided later on.

Task1. Classify Video (w.r.t. brightness, clearness and green tone levels).

Task2. Detect and Count Fish. (detect and count number of fish in each frame).

C.1 Part I (Efficiency – Full-automation *vs.* Manual)

a) Using the Intelligent Video Analyser, run Task1 for videos 1.mpeg – 10.mpeg. Use the **full automated** mode. You will be asked to choose a fast (using 3 frames) or slow (using 50 frames) option.
Do a fast task first followed by a slow one and view the result after each run (Option 1 in the End Menu). Note how many runs it took before you would select one of the fast and slow options.

b) Now perform Task1 **manually**. Please fill the classification values and the time taken (in minutes and seconds) in Table C.1.

Video	Brightness (dark/medium/bright)	Clearness (clear/medium/blur)	Green (green/not green)	Time (mins)
1.mpeg				
2.mpeg				
3.mpeg				
4.mpeg				
5.mpeg				
6.mpeg				
7.mpeg				
8.mpeg				
9.mpeg				
10.mpeg				

Table C.1: Manual time processing for video classification task.

Questions

- In general, there was no difference in the result between the slow and fast solutions. [Y/N]
  If Y after how many runs did you choose to run the fast option only? _____
- Solving the task using the automated method is less tedious than performing it manually [Y/N]
- Would you prefer to use the automated method if you have to perform this task frequently (*e.g.* on a daily basis)? [Y/N]

C.2 Part II (Learnability – Semi-automation)

This section concerns the pairs of videos given in Table C.2. For **each** pair, follow the instructions below.

c) Using Intelligent Video Analyser, perform Task2 for the first video of the pair using the **semi-automated** option. Using this option, you will be given a set of available tools to choose from for a particular subtask, if more than one can perform that subtask. When you are prompted to choose an image processing tool (*e.g. Create Background Model*) there will be descriptions and recommended conditions for each tool. A system recommended tool will also be provided. Try to make your selection based on the given recommendations.

d) View the result of applying this tool using the 'View Result' option provided in the menu. Repeat this task if you wish but by selecting a different tool. Option 6) in the menu will allow you to do this. Repeat as many times as you wish until you have identified the best tool that works for the video. Note this tool.

e) Now perform Task2 for the second video in the pair, again using the **semi-automated** option. If you can make a tool selection based on your experience with the results from the previous video, note this tool. Otherwise, note N/A.

Questions

- Having the option to provide domain information (constraints and video description) is appropriate to express the task in more detail [Y/N]
- Whenever you were prompted to make a tool selection, did you find the descriptions and recommendations useful? [Y/N]
- Did you find it easy to make changes to your requests (tasks, constraints and video descriptions)? [Y/N]

Pair	Video Name	Best Tool	Did you choose this Tool for the second video? [Y/N]
1	28.mpeg 17.mpeg		
2	13.mpeg 14.mpeg		
3	19.mpeg 20.mpeg		
4	27.mpeg 21.mpeg		
5	15.mpeg 16.mpeg		
6	1.mpeg 9.mpeg		
7	2.mpeg 10.mpeg		
8	28.mpeg 3.mpeg		
9	2.mpeg 7.mpeg		
10 11	13.mpeg 2.mpeg 19.mpeg 6.mpeg		
12	5.mpeg 4.mpeg		
13	28.mpeg 7.mpeg		
14	7.mpeg 8.mpeg		

Appendix D

ProcessLibrary.pl

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Process library for intelligent VIP system                                  %%
%% Gayathri Nadarajan                                                          %%
%% 30/05/08                                                                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- dynamic flag/2.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% method(Name, Precond, Decomp, Effects, Postcond).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% classify video
method(classify_video, [], [process_frames, perform_simple_classification,
write_frames_to_video], (nl), []).

% detect and count fish
method(detect_count_fish, [], [preliminary_analysis, process_each_frame_detection,
write_frames_to_video], (nl), []).

% classify video, detect and count fish
method(classify_video_detect_count_fish, [X =< Y], [preliminary_analysis, process_
each_frame, perform_classification, write_frames_to_video], (nl), []) :-
   frame_no(X),
   num_frames(Y).

% process_frames for classify video: recursive case
method(process_frames, [X =< Y], [grab_frame_image, compute_predominant_colours,
compute_main_texture_features, skip_frames, process_frames], (nl), []) :-
   frame_no(X),
```

```
   num_frames(Y).

% process_frames: base case -- do nothing when max no. frames reached
method(process_frames, [X > Y], [dummy], (nl), []) :-
   frame_no(X),
   num_frames(Y).

% process_each_frame_detection - recursive case
method(process_each_frame_detection, [X =< Y], [grab_frame_image, compute_pre-
dominant_colours, compute_main_texture_features, perform_detection, count_fish_
frame, compute_write_num_fish_frame, skip_frames, process_each_frame_detection],
(nl), []) :-
   frame_no(X),
   num_frames(Y).

% process_each_frame_detection - base case
method(process_each_frame_detection, [X > Y], [dummy], (nl), []) :-
   frame_no(X),
   num_frames(Y).

% process_each_frame tracking - recursive case
method(process_each_frame, [X =< Y], [grab_frame_image, compute_predominant_
colours, compute_main_texture_features, perform_detection, perform_tracking,
skip_frames, process_each_frame], (nl), []) :-
   frame_no(X),
   num_frames(Y).

% process_each_frame tracking - base case
method(process_each_frame, [X > Y], [dummy], (nl), []) :-
   frame_no(X),
   num_frames(Y).

% compute main texture features
method(compute_main_texture_features, [], [compute_histogram, compute_main_
statistical_moments], (nl), []).

% skip frames - skip half the number of frames for fast processing
method(skip_frames, [performance(processing_time)], [], ((frame_no(X), num_frames
(Y), X1 is X+Z, retract(frame_no(X)), assert(frame_no(X1)))), []) :- Z is Y/2.

% skip_frames -- go to next frame if it is not blocked
method(skip_frames, [blocked(0)], [], ((frame_no(X), X1 is X+1, retract(frame_
```

```
no(X)), assert(frame_no(X1)))), []) :-
   write('Going to next frame --> '),write(X1),nl.

% skip_frames -- skip 4 frames if big fish blocking screen
method(skip_frames, [blocked(1)], [], ((frame_no(X), X1 is X+4, retract(frame_
no(X)), assert(frame_no(X1)))), []) :-
   nl, write('Big fish blocking screen. Skipping 4 frames'),nl.

method(skip_frames, [quality(reliability)], [dummy], ((frame_no(X), X1 is X+1,
retract(frame_no(X)), assert(frame_no(X1)))), []).

% Perform detection
method(perform_detection, [frame_no(1)], [create_background_model, detect_objects_
blobs], (nl), []).

% Non-adaptive first frame, create bg model and detect blobs
method(detect_objects_blobs, [non_adaptive(true)], [detect_moving_objects,
detect_correct_blobs], (nl), []).

% Adaptive first frame, skip straight to detect and count
method(detect_objects_blobs, [non_adaptive(false)], [dummy], (nl), []).

% For non-adaptive non-first frames, recreate bg model
method(perform_detection, [non_adaptive(true)], [create_background_model,
detect_moving_objects, detect_correct_blobs], (nl), []).

% MA model non-first frame, update bg model
method(perform_detection, [background_model(moving_average)], [update_moving_
average_model, detect_moving_objects, detect_correct_blobs], (nl), []).

% For adaptive models, bg model does not need to be updated.
method(perform_detection, [], [dummy], (nl), []).

% If no single background model is suitable, fuse GMM and MA
method(create_background_model, [], [create_gaussian_mixture_model,
create_moving_average_model, fuse_bg_images], (nl), []).

% Perform Tracking
method(perform_tracking, [], [tracking_initialisation, compute_connected_compo-
nents, compute_blob_features, compute_closest_blob_rec, compute_write_num_fish_
frame, compute_write_num_fish_video, compute_fish_blocking_screen], (nl), []).
```

```
method(tracking_initialisation, [], [extract_hsv_values, compute_backprojection],
(nl), []).

method(compute_blob_features, [X > 10], [compute_camshift],
(assert(num_segments(10))), []) :-
   frame_no(X).

method(compute_blob_features, [], [compute_camshift], (assert(num_segments(Y))),
[]) :-
   frame_no(X),
   Y is X-1.

method(compute_closest_blob_rec, [num_segments(0)], [dummy], (nl), []) :-
   write('Skipping compute closest blob, num segments 0'), nl, !.

method(compute_closest_blob_rec, [], [compute_closest_blob, compute_closest_blob_
rec], (nl), [Y >= 0]) :-
   retract(num_segments(X)),
   Y is X-1,
   assert(num_segments(Y)).

method(perform_simple_classification, [], [compute_write_luminosity,
compute_write_clearness, compute_write_presence_algae], (nl), []).

method(perform_classification, [], [compute_write_luminosity, compute_write_clear-
ness, compute_write_presence_algae, compute_write_presence_fish], (nl), []).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 0. Primitive tasks: primitive/5 - determines primitive      %%
% tasks from list of independent executables                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
primitive(PP, Input, Preconds, Add_list_file, Postconds) :-
   independent_executable(PP, Fn_call, Preconds, Add_list_file, Input, Output,
   Postconds),
   write('Independent executable '), write(PP), write(' found '),nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% independent_executable(User_terminology, Fn_call, Preconds_list, %%
%% Assert_list_file, Input_list, Output_list, Postconds_list)       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1. View Video videos/1.mpeg 50
independent_executable(view_video, view_video, [], [], [Filename, Num_Frames],
```

```
[Displayed_video, Frame_rate, No_frames, Format, Date, Bg_images], []) :-
   input_file(File), atom_concat('videos/', File, Filename),
   num_frames(Num_Frames).

%% 2. Preliminary Analysis videos/1.mpeg
independent_executable(preliminary_analysis, preliminary_analysis, [], [Result_
file], [Filename], [Init_brightness, Init_clearness, Init_speed, Init_texture,
Percent_bg_obj, Noise_type], []) :-
   input_file(File), atom_concat('videos/', File, Filename),
   atom_concat('Prelim_', File, File1),
   atom_concat(File1, '.dat', Result_file).

%% 3. Grab Frame Image videos/1.mpeg
independent_executable(grab_frame_image, grab_frame_image, [], [], [Filename,
Fr_no], [Current_frame_image], []) :-
   input_file(File), atom_concat('videos/', File, Filename), frame_no(Fr_no).

%% 4. Extract RGB Colours Video_1.mpeg/2
independent_executable(extract_RGB_colours, extract_RGB_colours, [], [],
[Current_frame_image], [Red_image, Green_image, Blue_image, Yellow_image], []) :-
   input_file(I),
   frame_no(Fno), number_chars(Fno,Fno1),atom_chars(Fnum,Fno1),
   atom_concat('Video_',I,X),
   atom_concat(X,'/',Y), atom_concat(Y,Fnum,Current_frame_image).

%% 4. Compute Predominant Colours Video_1.mpeg/2
independent_executable(compute_predominant_colours, compute_predominant_colours,
[], [], [Current_frame_image], [Red_image, Green_image, Blue_image, Yellow_image],
[]) :-
   input_file(I),
   frame_no(Fno), number_chars(Fno,Fno1),atom_chars(Fnum,Fno1),
   atom_concat('Video_',I,X),
   atom_concat(X,'/',Y), atom_concat(Y,Fnum,Current_frame_image).

%% 5. Compute Histogram Video_1.mpeg/2
independent_executable(compute_histogram, compute_histogram, [], [], [Current_
frame_image], [Histogram_value, Histogram_image], []) :-
          input_file(I), frame_no(Fno),
          number_chars(Fno,Fno1), atom_chars(Fnum,Fno1),
          atom_concat('Video_',I,X),
          atom_concat(X,'/',Y), atom_concat(Y,Fnum,Current_frame_image).
```

```
%% 6. Compute Main Statistical Moments Video_1.mpeg/2/Hist_Array_2.dat
independent_executable(compute_main_statistical_moments, compute_main_statisti-
cal_moments, [], [], [Hist_file], [Mean, Variance, Third_moment, Fourth_moment,
Entropy, Uniformity, Smoothness], []) :-
   input_file(I), frame_no(Fno),
   number_chars(Fno,Fno1), atom_chars(Fnum,Fno1),
   atom_concat('Video_',I,X),
   atom_concat(X,'/',Y), atom_concat(Y,Fnum,Current_frame_image),
   atom_concat(Current_frame_image,'/Hist_Array_',H1),
   atom_concat(H1,Fnum,H2), atom_concat(H2,'.dat',Hist_file).

%% 7. Compute Gabor Filter Video_1.mpeg/2.jpg Video_1.mpeg/ (3 45)
independent_executable(compute_Gabor_filter, compute_Gabor_filter, [], [],
[Current_frame_image, V_dir], [Gabor_filter], []) :-
   retrieve_video_dir(V_dir),
   retrieve_frame_image(Current_frame_image).

%% 8a. Create Background Model
independent_executable(create_background_model, create_background_model, [],
[], [], [], []).

%% 8. Create Gaussian Background Model Video_1.mpeg/BackgroundImages/
independent_executable(create_gaussian_bg_model, create_gaussian_bg_model, [],
['gaussian.dat'],[Directory],[Bg_image1, Bg_image2],[background_model(gaussian)]):-
   retrieve_video_dir(V_dir),
   atom_concat(V_dir,'BackgroundImages/',Directory).

%% 9. Create Gaussian Mixture Model videos/1.mpeg Video_1.mpeg/GMM/ 50
independent_executable(create_gaussian_mixture_model, create_gaussian_mixture_model,
[], ['gaussian_mixture.dat'], [Filename, V_dir, NumFrames], [Bg_image],
[background_model(gaussian_mixture)]) :-
   input_file(File), atom_concat('videos/', File, Filename),
   retrieve_video_dir(V_dir),
   atom_concat(V_dir,'GMM/',GMM_dir),
   num_frames(NumFrames).

%% 10. Create Moving Average Model Video_1.mpeg/3.jpg Video_1.mpeg/Moving_Average/
0.020independent_executable(create_moving_average_model, create_moving_average_model,
[], ['moving_average.dat'], [Frame_image, MA_dir, Learning_speed], [Bg_image],
[background_model(moving_average)]) :-
   retrieve_frame_image(Frame_image),
```

```
   retrieve_video_dir(V_dir),
   atom_concat(V_dir,'Moving_Average/',MA_dir),
   Learning_speed = '0.020'.

%% 11. Create Intra-Frame Difference Model Video_1.mpeg/ Video_1.mpeg/IntraFrame-
Difference/
independent_executable(create_intra_frame_difference_model, create_intra_frame_
model, [], ['intra_frame.dat'], [Dir, Output_dir], [Bg_matrix1, Bg_matrix2],
[background_model(ifd)]) :-
   retrieve_video_dir(Dir),
   atom_concat(Dir,'IntraFrameDifference/',Output_dir).

%% 12. Create W4 Model Video_1.mpeg/(Images/) Video_1.mpeg/W4/
independent_executable(create_w4_model, create_w4_model, [], ['w4.dat'],
[Dir, W4_dir], [Bg_matrix1, Bg_matrix2, Bg_matrix3], [background_model(w4)]) :-
   retrieve_video_dir(Dir),
   atom_concat(Dir,'W4/',W4_dir).

%% 13. Create Poisson Model Video_1.mpeg/Images/ Video_1.mpeg/Poisson/
independent_executable(create_poisson_model, create_poisson_model, [], ['poisson.
dat'], [Dir, Bg_dir], [Bg_matrix1, Bg_matrix2], [background_model(poisson)]) :-
   retrieve_video_dir(Dir),
   atom_concat(Dir,'Poisson/',Bg_dir).

%% 14. Create Adaptive Poisson Model videos/1.mpeg Video_1.mpeg/
independent_executable(create_adaptive_poisson_model, create_adaptive_poisson_model,
[background_model(adaptive_poisson)]) :-
   input_file(I),
   atom_concat('videos/', I, Video),
   retrieve_video_dir(V_dir).

%% 15. Update Moving Average Background Model Video_1.mpeg/2.jpg
independent_executable(update_moving_average_model, update_moving_average_model, [],
[], [Frame_image, Old_bg_image, Learning_speed], [Updated_bg_image], []) :-
   retrieve_frame_image(Frame_image),
   retrieve_video_dir(V_dir),
   atom_concat(V_dir,'Moving_Average/Background.jpg',Old_bg_image),
   Learning_speed = '0.020'.

%% 16. Fuse BG Images
independent_executable(fuse_bg_images, fuse_bg_images, [], [], [GMM_model, MA_model,
F_Dir], [Fused_image], []) :-
```

```
   retrieve_video_dir(V_dir),
   atom_concat(V_dir, 'GMM/Background.jpeg', GMM_model),
   atom_concat(V_dir, 'Moving_Average/Background.jpg',MA_model),
   retrieve_frame_dir(F_dir).

%% 17. Detect Moving Objects 2 Video_1.mpeg/IntraFrameDifference/intraFrame.jpg
Video_1.mpeg/2.jpg Video_1.mpeg/2/
independent_executable(detect_moving_objects, detect_moving_objects, [], [],
[Bg_model_type, Bg_frame_image, Frame_image, F_dir], [BW_image_with_blobs], []) :-
   get_bg_model(Bg_model_type),
   retrieve_frame_dir(F_dir),
   retrieve_frame_image(Frame_image),
   load_bg_model_image(Bg_model_type, Bg_frame_image).

%% 18. Perform Morphological Operation Video_1.mpeg/2/IntraFrameDifference/Binary_
Image.jpeg Video_1.mpeg/2/
independent_executable(detect_correct_blobs, detect_correct_blobs, [], [],
[BW_image_with_blobs, F_dir], [Img_potential_correct_blobs], []) :-
   get_bg_model(T),
   retrieve_frame_dir(F_dir),
   atom_concat(F_dir, 'Binary_Image.jpeg', BW_image_with_blobs).

%% 19. Extract HSV Values Video_1.mpeg/2.jpg Video_1.mpeg/2/
independent_executable(extract_hsv_values, extract_hsv_values, [], [], [F_image,
F_dir], [Hue_image, Saturation_image, Value_image], []) :-
   retrieve_frame_image(F_image),
   retrieve_frame_dir(F_dir).

%% 20. Compute Backprojection Video_1.mpeg/2/Hue.jpg Video_1.mpeg/2/
independent_executable(compute_backprojection, compute_backprojection, [], [],
[Hue_image, F_dir], [Hue_backprojection_image], []) :-
   retrieve_frame_dir(F_dir),
   atom_concat(F_dir, 'Hue.jpg', Hue_image).

%% 21. Compute Connected Components Video_1.mpeg/2/CorrectBlobs.jpeg Video_1.mpeg/2/
independent_executable(compute_connected_components, compute_connected_components,
[], [], [BW_image_with_blobs, F_dir], [Image_with_blobs, Num_blobs], []) :-
   retrieve_frame_dir(F_dir),
   atom_concat(F_dir, 'CorrectBlobs.jpeg', BW_image_with_blobs).

%% 22. Compute Camshift and Ratio Video_1.mpeg/2.jpg Video_1.mpeg/2/BlobsBW.jpeg
Video_1.mpeg/2/
```

```
independent_executable(compute_camshift, compute_camshift, [], [],
[F_img, Image_with_blobs, F_dir], [Centre, Orientation, Size], []) :-
   retrieve_frame_dir(F_dir),
   retrieve_frame_image(F_img),
   atom_concat(F_dir,'BlobsBW.jpeg', Image_with_blobs).

%% 23. Compute Closest Blob: Video_1.mpeg/1/ Video_1.mpeg/2/
independent_executable(compute_closest_blob, compute_closest_blob, [], [], [F_dir1,
F_dir2, Num_blobs1, Num_blobs2], [Minimum_euclidian_distance], []) :-
   retrieve_frame_dir(F_dir1),
   num_segments(Seg),
   frame_no(X), Y is X-Seg,
   retrieve_video_dir(V_dir),
   atom_concat(V_dir,Y, Y1),
   atom_concat(Y1, '/', F_dir2).

%% 24. Compute and Write Number of Fish in Frame and Video: Video_1.mpeg/2/
Video_1.mpeg/2fish.dat Video_1.mpeg/2.jpg Video_1.mpeg/ 2.jpg (true)
independent_executable(compute_write_num_fish_frame, count_write_num_fish_frame,
[], [], [F_dir, Blob_file, F_img, V_dir, F_name], [Num_fish_frame, Blob_counted,
Final_frame_image], []) :-
   retrieve_frame_dir(F_dir),
   retrieve_frame_image(F_img),
   frame_no(X), number_chars(X,XC), atom_chars(XA,XC),
   retrieve_video_dir(V_dir),
   atom_concat(V_dir, XA, FXA), atom_concat(FXA,'fish.dat', Blob_file),
   atom_concat(XA,'.jpg',F_name).

%% 25. Determine Presence of Fish Blocking Screen Video_1.mpeg/3/CorrectBlobs.jpeg
Video_1.mpeg/3/
independent_executable(compute_fish_blocking_screen, compute_fish_blocking_screen,
[], [Screen_blocked_file], [Image_with_blob, F_dir], [Presence], []):-
   retrieve_frame_dir(F_dir),
   atom_concat(F_dir, 'CorrectBlobs.jpeg', Image_with_blob),
   atom_concat(F_dir, 'block.dat', Screen_blocked_file).

%% 26. Compute and Write Average Luminosity Video_1.mpeg/
independent_executable(compute_write_luminosity, compute_write_luminosity, [],
['brightness.dat'], [V_dir], [Final_frame_image], []) :-
   retrieve_video_dir(V_dir).

%% 27. Compute and Write Average Clearness Video_1.mpeg/
```

```
independent_executable(compute_write_clearness, compute_write_clearness, [],
['clearness.dat'], [V_dir], [Final_frame_image], []) :-
   retrieve_video_dir(V_dir).

%% 28. Compute and Write Presence of Fish Video_1.mpeg/
independent_executable(compute_write_presence_fish, compute_write_presence_fish,
[], [], [V_dir], [Final_frame], []) :-
   retrieve_video_dir(V_dir).

%% 29. Compute and Write Presence of Algae Video_1.mpeg/
independent_executable(compute_write_presence_algae, compute_write_presence_algae,
[], ['green.dat'], [V_dir], [Final_image], []) :-
   retrieve_video_dir(V_dir).

%% 30. Write to (Final) Output Video Final_1.mpeg.avi Video_1.mpeg/Output/
independent_executable(write_frames_to_video, write_frames_to_video, [], [],
[Final_video, Images_dir], [], []) :-
   input_file(I),
   atom_concat('Final_', I, FV),
   atom_concat(FV, '.avi', Final_video),
   retrieve_video_dir(V_dir),
   atom_concat(V_dir, 'Output/', Images_dir).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% flags - initially to show that tools for primitive processes
%% have not been selected, checked to true when used.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flag(view_video,false).
flag(preliminary_analysis,false).
flag(grab_frame_image,false).
flag(compute_predominant_colours,false).
flag(compute_histogram,false).
flag(compute_main_statistical_moments,false).
flag(compute_Gabor_filter,false).
flag(create_background_model,false).
flag(update_moving_average_model,false).
flag(fuse_bg_images,false).
flag(detect_moving_objects,false).
flag(detect_correct_blobs,false).
flag(perform_morphological_operation,false).
flag(compute_hsv_values,false).
flag(compute_backprojection,false).
```

```
flag(compute_connected_components,false).
flag(compute_ratio_convex_hull_blob,false).
flag(compute_camshift,false).
flag(compute_closest_blob,false).
flag(detect_track,false).
flag(count_fish_frame,false).
flag(compute_write_num_fish_frame,false).
flag(compute_write_num_fish_video,false).
flag(compute_fish_blocking_screen,false).
flag(compute_write_luminosity,false).
flag(compute_write_clearness,false).
flag(compute_write_presence_fish,false).
flag(compute_write_presence_algae,false).
flag(write_frames_to_video,false).
```

Appendix E

GoalOntology.pl

```
:- multifile class/1, instance_of/2, subclass_of/2, class_rel/3.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Ontology notation for goal and constraints    %%
%% for video and image processing tasks          %%
%%           Gaya Nadarajan, 2008-10              %%
%%               gaya.n@ed.ac.uk                  %%
%%                                                %%
%% Reference:                                     %%
%% 1. FBPML Data Description Language             %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% class(Class_name).                            %%
%% Classes in the domain                         %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class(thing).

class(goal).
class(compression).

class(segmentation).

class(detection).
class(detect_presence).
class(count_occurrence).

class(classification).
class(classify_object).
class(classify_image_video).
```

```
class(constraint_category).
class(control_constraint).
class(feedback_constraint).
class(regulation_constraint).

class(constraint_descriptor).

class(control_constraint_descriptor).
class(performance_criteria).

class(feedback_constraint_descriptor).

class(regulation_constraint_descriptor).

class(constraint_qualifier).

class(acceptable_error).
class(separability_acceptable_error).
class(on_boundary).
class(segmentation_acceptable_error).
class(accuracy).

class(criteria_to_optimise).

class(detail_level).
class(segmentation_detail_level).
class(boundaries).
class(occurrences).
class(separability_detail_level).
class(pixels).
class(region).

class(excluded_included_element).
class(quality_criteria).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% subclass_of(Child, Parent).                                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subclass_of(goal, thing).

subclass_of(compression, goal).
```

```
subclass_of(segmentation, goal).
subclass_of(detection, goal).
subclass_of(classification, goal).
subclass_of(display, goal).

subclass_of(detect_presence, detection).
subclass_of(count_occurrence, detection).

subclass_of(classify_object, classification).
subclass_of(classify_image_video, classification).

subclass_of(control_constraint, constraint_category).
subclass_of(feedback_constraint, constraint_category).
subclass_of(regulation_constraint, constraint_category).

subclass_of(control_constraint_descriptor, constraint_descriptor).
subclass_of(feedback_constraint_descriptor, constraint_descriptor).
subclass_of(regulation_constraint_descriptor, constraint_descriptor).

subclass_of(performance_criteria, constraint_qualifier).

subclass_of(acceptable_error, constraint_qualifier).
subclass_of(criteria_to_optimise, constraint_qualifier).
subclass_of(detail_level, constraint_qualifier).
subclass_of(excluded_included_element, constraint_qualifier).
subclass_of(quality_criteria, constraint_qualifier).

subclass_of(separability_acceptable_error, acceptable_error).
subclass_of(on_boundary, acceptable_error).
subclass_of(separability_acceptable_error, acceptable_error).
subclass_of(accuracy, acceptable_error).

subclass_of(segmentation_detail_level, detail_level).
subclass_of(boundaries, detail_level).
subclass_of(occurrences, detail_level).
subclass_of(separability_detail_level, detail_level).
subclass_of(pixels, detail_level).
subclass_of(region, detail_level).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% class_rel(Rel, Class_1, Class_2)                                          %%
%% Relationship between two classes                                          %%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class_rel(hasQualityCriteria, control_constraint, quality_criteria).
class_rel(hasPerformanceCriteria, performance_criteria).
class_rel(hasDescriptor, constraint_category, constraint_descriptor).
class_rel(hasConstraintCategory, goal, constraint_category).
class_rel(hasConstraintQualifier, goal, constraint_qualifier).

class_rel(is_related_to, accuracy, detection).
class-rel(is_related_to, occurrence, detection).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The instance_of(Instance, Class) predicate    %%
%% stores the instance and its type (Class).     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
instance_of(best_compromise, quality_criteria).
instance_of(reliability, quality_criteria).
instance_of(robustness, quality_criteria).

instance_of(memory, performance_criteria).
instance_of(processing_time, performance_criteria).

instance_of(extended_element_descriptor, feedback_constraint_descriptor).
instance_of(included_element_descriptor, feedback_constraint_descriptor).

instance_of(prefer_separate, separability_acceptable_error).
instance_of(prefer_not_separate, separability_acceptable_error).

instance_of(prefer_boundaries_inside_the_region, on_boundary).
instance_of(prefer_region_outside_the_region, on_boundary).

instance_of(prefer_sub_segmentation, segmentation_acceptable_error).
instance_of(prefer_over_segmentation, segmentation_acceptable_error).

instance_of(prefer_false_alarm_than_miss, accuracy).
instance_of(prefer_miss_than_false_alarm, accuracy).

instance_of(over_segmentation, segmentation_detail_level).
instance_of(sub_segmentation, segmentation_detail_level).

instance_of(boundaries_outside_the_region, boundaries).
instance_of(boundaries_inside_the_region, boundaries).
```

```
instance_of(all, occurrences).
instance_of(at_least_one, occurrences).

instance_of(separate_just_touching_and_not_aggregated, separability_detail_level).
instance_of(do_not_separate, separability_detail_level).

instance_of(all_pixels, pixels).
instance_of(at_least_one_pixel, pixels).

instance_of(one_object_per_region, region).
instance_of(contour_regularisation, region).

instance_of(compress_without_loss, compression).
instance_of(compress_with_loss, compression).

instance_of(partition, segmentation).
instance_of(extraction, segmentation).
instance_of(eliminate, segmentation).

instance_of(detect_presence_fish, detect_presence).
instance_of(detect_presence_coral, detect_presence).
instance_of(count_fish, count_occurrence).

instance_of(view_video, display).

instance_of(classify_fish_green_chromis, classify_object).
instance_of(classify_video, classify_image_video).
```

Appendix F

VideoDescriptionOntology.pl

```
:- multifile class/1, subclass_of/2, instance_of/2, class_rel/3.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Ontology notation for video description       %%
%% for video and image processing tasks          %%
%%           Gaya Nadarajan, 2008-9               %%
%%               gaya.n@ed.ac.uk                  %%
%%                                                %%
%% Reference:                                     %%
%% 1. FBPML Data Description Language             %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% class(Class_name).                             %%
%% Classes in the domain                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class(thing).

class(video_description).

class(video_image_class_thing).

class(description_element).
class(acquisition_element).
class(visual_primitive).

class(descriptor).

class(acquisition_condition).
class(lighting).
```

```
class(clearness).
class(green_tone).

class(boundary).

class(colour).
class(luminosity).
class(colour_model).
class(saturation).
class(hue).

class(edge).

class(geometric).
class(orientation).
class(position).
class(size).

class(photometric).

class(shape).

class(texture).

class(resulting_image).
class(detect_descriptor).
class(format_descriptor).
class(noise_descriptor).

class(observation).

class(observation_category).
class(acquisition_system).
class(business_concept).
class(visual_rendering).

class(observation_category).
class(physical).
class(perceptive).
class(semantic).
```

```
class(value).

class(value_type).
class(qualifier_type).
class(quantifier_type).

class(relation).
class(spatial_relation).
class(temporal_relation).
class(value_relation).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% subclass_of(Child, Parent).                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subclass_of(video_description, thing).

subclass_of(video_image_class_thing, video_description).
subclass_of(value, video_description).
subclass_of(value_type, video_description).
subclass_of(relation, video_description).

subclass_of(description_element, video_image_class_thing).
subclass_of(descriptor, video_image_class_thing).
subclass_of(observation, video_image_class_thing).
subclass_of(observation_category, video_image_class_thing).
subclass_of(observation_level, video_image_class_thing).

subclass_of(acquisition_element, description_element).
subclass_of(visual_primitive, description_element).

subclass_of(acquisition_condition, descriptor).
subclass_of(lighting, acquisition_condition).
subclass_of(clearness, acquisition_condition).

subclass_of(boundary, descriptor).

subclass_of(colour, descriptor).
subclass_of(luminosity, colour).
subclass_of(colour_model, colour).
subclass_of(saturation, colour).
subclass_of(hue, colour).
subclass_of(green_tone,colour).
```

```
subclass_of(edge, descriptor).

subclass_of(geometric, descriptor).
subclass_of(orientation, geometric).
subclass_of(position, geometric).
subclass_of(size, geometric).

subclass_of(photometric, descriptor).
subclass_of(shape, descriptor).
subclass_of(texture, descriptor).

subclass_of(resulting_image, descriptor).
subclass_of(defect_descriptor, resulting_image).
subclass_of(format_descriptor, resulting_image).
subclass_of(noise_descriptor, resulting_image).

subclass_of(acquisition_system, observation_category).
subclass_of(business_concept, observation_category).
subclass_of(visual_rendering, observation_category).

subclass_of(physical, observation_level).
subclass_of(perceptive, observation_level).
subclass_of(semantic, observation_level).

subclass_of(qualifier_type, value_type).
subclass_of(quantifier_type, value_type).

subclass_of(spatial_relation, relation).
subclass_of(temporal_relation, relation).
subclass_of(value_relation, relation).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The instance_of(Instance, Class) predicate    %%
%% stores the instance and its type (Class).     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
instance_of(clear, clearness).
instance_if(medium, clearness).
instance_of(blur, clearness).

instance_of(dark, luminosity).
instance_of(medium, luminosity).
```

```
instance_of(bright, luminosity).

instance_of(low, saturation).
instance_of(medium, saturation).
instance_of(high, saturation).

instance_of(low, hue).
instance_of(medium, hue).
instance_of(high, hue).

instance_of(small, size).
instance_of(medium, size).
instance_of(big, size).

instance_of(green, green_tone).
instance_of(not_green, green_tone).

instance_of(above, spatial_relation).
instance_of(below, spatial_relation).
instance_of(to_the_left, spatial_relation).
instance_of(to_the_right, spatial_relation).
instance_of(in_included_in, spatial_relation).
instance_of(is_disconnected_from, spatial_relation).
instance_of(is_externally_connected, spatial_relation).
instance_of(is_overlapping, spatial_relation).

instance_of(date, temporal_relation).
instance_of(speed, temporal_relation).
instance_of(comparison_relation, value_relation).
instance_of(logical_relation, value_relation).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% class_rel(Rel, Class_1, Class_2)              %%
%% Relationship between two classes              %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class_rel(hasDescriptionElement, observation_category, description_element).
class_rel(hasValueType, descriptor, value_type).
class_rel(hasPhysicalPart, observation, physical).
class_rel(hasPerceptivePart, observation, perceptive).
class_rel(hasSemanticPart, observation, semantic).
```

Appendix G

CapabilityOntology.pl

```
:- multifile class/1, instance_of/2, subclass_of/2, instance_att_list/2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Ontology notation for capabilities                                  %%
%% for video and image processing tasks                                %%
%%           Gaya Nadarajan, 2007-10                                    %%
%%               gaya.n@ed.ac.uk                                        %%
%%                                                                      %%
%% Reference:                                                           %%
%% 1. FBPML Data Description Language                                   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% class(Class_name).                                                   %%
%% Classes in the domain                                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class(thing).

class(tool).
class(capability_thing).
class(performance_criteria).

class(vip_operation).

class(image_acquisition).
class(image_transforms).
class(image_analysis).
class(video_analysis).
class(classification).
```

```
class(image_formation_preprocessing).
class(point).
class(geometric).
class(domain).
class(extent).

class(colour).
class(feature_extraction).
class(detection).
class(tracking).

class(max_likelihood).
class(thresholding).
class(neural_networks).
class(video_classification).



class(classification).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% subclass_of(Child, Parent).                                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subclass_of(tool, thing).
subclass_of(capability_thing, thing).
subclass_of(performance_criteria, thing).

subclass_of(vip_operation, capability_thing).

subclass_of(image_acquisition, vip_operation).
subclass_of(image_transforms, vip_operation).
subclass_of(image_analysis, vip_operation).
subclass_of(video_analysis, vip_operation).
subclass_of(classification, vip_operation).

subclass_of(geometric, vip_operation).
subclass_of(translation, geometric).
subclass_of(scaling, geometric).
subclass_of(rotation, geometric).
subclass_of(shearing, geometric).
subclass_of(transposing, geometric).
subclass_of(transformation, geometric).
```

```
subclass_of(domain, vip_operation).
subclass_of(fourier_transform, domain).

subclass_of(extent, vip_operation).
subclass_of(cropping, extent).class(classification).
subclass_of(mosaicing, extent).
subclass_of(tiling, extent).
subclass_of(region_of_interest, extent).
basic_structures_operations_tool
subclass_of(colour, image_analysis).
subclass_of(feature_extraction, image_analysis).
subclass_of(detection, image_analysis).
subclass_of(tracking, image_analysis).

subclass_of(max_likelihood, classification).
subclass_of(thresholding, classification).
subclass_of(neural_networks, classification).
subclass_of(video_classification, classification).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The instance_of(Instance, Class) predicate    %%
%% stores the instance and its type (Class).     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Performance Criteria
instance_of('Clear and Fast Background Movement', performance_criteria).
instance_of('Fast Background Movement and Blur', performance_criteria).
instance_of('Not Uniform, Low Background Movement and Few Fish',
performance_criteria).
instance_of('Not Uniform, Fast Background Movement and More Accurate',
performance_criteria).
instance_of('Uniform, Low Background Movement and Less Accurate',
performance_criteria).
instance_of('Uniform, Blur and Fast Background Movement', performance_criteria).
instance_of('Uniform, Low Background Movement and Fast Performance',
performance_criteria).

%% VIP tools
instance_of(motion_analysis_object_tracking_tool, tool).
instance_of(image_analysis_tool, tool).
instance_of(structural_analysis_tool, tool).
```

```
instance_of(object_recognition_tool, tool).
instance_of('3D_reconstruction_tool', tool).
instance_of(basic_structures_operations_tool, tool).

%% VIP Operations
instance_of(quantitative_visualisation, image_formation_preprocessing).
instance_of(preprocessing_initialisation, image_formation_preprocessing).

instance_of(arithmetic, point).
instance_of(logical, point).
instance_of(functions, point).

instance_of(translation, geometric).
instance_of(scaling, geometric).
instance_of(rotation, geometric).
instance_of(shearing, geometric).
instance_of(transposing, geometric).
instance_of(transformation, geometric).

instance_of(fourier_transform, domain).

instance_of(cropping, extent).
instance_of(mosaicing, extent).
instance_of(tiling, extent).
instance_of(region_of_interest, extent).

instance_of(compute_predominant_colours, colour).

instance_of(edge_detection, feature_extraction).
instance_of(morphology, feature_extraction).
instance_of(compute_main_texture_features, feature_extraction).

instance_of(create_background_model, detection).
instance_of(update_background_model, detection).
instance_of(detect_correct_blobs, detection).
instance_of(count_fish_frame, detection).

instance_of(tracking_initialisation, tracking).
instance_of(compute_blob_features, tracking).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% hasDescription(Tool, Description)                                        %%
```

```
%% contains description of a tool                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hasDescription(create_background_model, ‘creates an image that represents the
background so that objects on the current frame image can be detected.’).
hasDescription(_,’’).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% instance_att_list(Description, List)          %%
%% A list of domain descriptions, List that fit  %%
%% the description, Description                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gaussian Background Model
instance_att_list(‘Clear and Fast Background Movement’, [clearness(high),
bg_movement(high)]).

% Adaptive Gaussian Mixture Model
instance_att_list(‘Fast Background Movement and Blur’, [not(uniformity(high)),
not(clearness(high)), bg_movement(high)]).

% Adaptive Poisson Model
instance_att_list(‘Uniform, Blur and Fast Background Movement’, [uniformity(high),
not(clearness(high)), bg_movement(high)]).

% Moving Average Model
instance_att_list(‘Uniform, Low Background Movement and Less Accurate’,
[accuracy(prefer_false_alarm_than_miss), uniformity(high), bg_movement(low)]).

% Poisson Model
instance_att_list(‘Uniform, Low Background Movement and Fast Performance’,
[uniformity(high), bg_movement(low), performance(processing_time)]).

% Intra-Frame Difference Model
instance_att_list(‘Not Uniform, Low Background Movement and Few Fish’,
[percentage_bg_object(high),not(bg_movement(high)),not(uniformity(high))]).

% W4 Model
instance_att_list(‘Not Uniform, Fast Background Movement and More Accurate’,
[not(uniformity(high)),bg_movement(high),accuracy(prefer_miss_than_false_alarm)]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% hasPerformanceIndicator(Instance, Description) %%
```

```
%% A tool instance and its suitable description   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hasPerformanceIndicator(create_gaussian_bg_model,‘Clear and Fast Background
Movement’).
hasPerformanceIndicator(create_gaussian_mixture_model,‘Fast Background Movement
and Blur’).
hasPerformanceIndicator(create_adaptive_poisson_model, ‘Uniform, Blur and Fast
Background Movement’).
hasPerformanceIndicator(create_moving_average_model,‘Uniform, Low Background
Movement and Less Accurate’).
hasPerformanceIndicator(create_poisson_model, ‘Uniform, Low Background Movement
and Fast Performance’).
hasPerformanceIndicator(create_intra_frame_difference_model,‘Not Uniform, Low
Background Movement and Few Fish’).
hasPerformanceIndicator(create_w4_model,‘Not Uniform, Fast Background Movement
and More Accurate’).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% specialisation_of(Instance1, Instance2)       %%
%% An instance that is more specialised than     %%
%% a root instance, Instance2                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
specialisation_of(cvAcc, motion_analysis_object_tracking_tool).
specialisation_of(cvSquareAcc, motion_analysis_object_tracking_tool).
specialisation_of(cvMultiplyAcc, motion_analysis_object_tracking_tool).
specialisation_of(cvRunningAvg, motion_analysis_object_tracking_tool).
specialisation_of(cvUpdateMotionHistory, motion_analysis_object_tracking_tool).
specialisation_of(cvCalcMotionGradient, motion_analysis_object_tracking_tool).
specialisation_of(cvCalcGlobalOrientation, motion_analysis_object_tracking_tool).
specialisation_of(cvSegmentMotion, motion_analysis_object_tracking_tool).
specialisation_of(cvCamShift, motion_analysis_object_tracking_tool).
specialisation_of(cvMeanShift, motion_analysis_object_tracking_tool).
specialisation_of(cvSnakeImage, motion_analysis_object_tracking_tool).
specialisation_of(cvCalcOpticalFlowHS, motion_analysis_object_tracking_tool).
specialisation_of(cvCreateKalman, motion_analysis_object_tracking_tool).
specialisation_of(cvConDensation, motion_analysis_object_tracking_tool).

specialisation_of(extract_hsv_values, motion_analysis_object_tracking_tool).
specialisation_of(compute_backprojection, motion_analysis_object_tracking_tool).
specialisation_of(compute_connected_components, motion_analysis_object_tracking_
tool).
specialisation_of(compute_ratio_convex_hull_blob, motion_analysis_object_tracking_
```

```
tool).
specialisation_of(compute_camshift, motion_analysis_object_tracking_tool).
specialisation_of(compute_closest_blob, motion_analysis_object_tracking_tool).
specialisation_of(detect_track, motion_analysis_object_tracking_tool).
specialisation_of(compute_write_num_fish_frame, motion_analysis_object_tracking_
tool).
specialisation_of(compute_write_num_fish_video, motion_analysis_object_tracking_
tool).
specialisation_of(compute_fish_blocking_screen, motion_analysis_object_tracking_
tool).

specialisation_of(cvFindContours, image_analysis_tool).
specialisation_of(cvLaplace, image_analysis_tool).
specialisation_of(cvSobel, image_analysis_tool).
specialisation_of(cvCanny, image_analysis_tool).
specialisation_of(cvPreCornerDetect, image_analysis_tool).
specialisation_of(cvCornerEigenValsAndVecs, image_analysis_tool).
specialisation_of(cvGoodFeaturesToTrack, image_analysis_tool).
specialisation_of(cvHoughLines, image_analysis_tool).
specialisation_of(cvSumPixels, image_analysis_tool).
specialisation_of(cvMean_StdDev, image_analysis_tool).
specialisation_of(cvNorm, image_analysis_tool).
specialisation_of(cvMoments, image_analysis_tool).
specialisation_of(cvPyrSegmentation, image_analysis_tool).
specialisation_of(cvStructuringElementEx, image_analysis_tool).
specialisation_of(cvErode, image_analysis_tool).
specialisation_of(cvDilate, image_analysis_tool).
specialisation_of(cvDistTransform, image_analysis_tool).
specialisation_of(cvAdaptiveThreshold, image_analysis_tool).
specialisation_of(cvCreateHist, image_analysis_tool).
specialisation_of(cvCalcBackProject, image_analysis_tool).
specialisation_of(cvConnectedComp, image_analysis_tool).
specialisation_of(cvHistogram, image_analysis_tool).

specialisation_of(extract_RGB_colours, image_analysis_tool).
specialisation_of(compute_histogram, image_analysis_tool).
specialisation_of(compute_main_statistical_moments, image_analysis_tool).
specialisation_of(compute_gabor_filter, image_analysis_tool).
specialisation_of(create_gaussian_bg_model, image_analysis_tool).
specialisation_of(create_gaussian_mixture_model, image_analysis_tool).
specialisation_of(create_moving_average_model, image_analysis_tool).
specialisation_of(create_intra_frame_difference_model, image_analysis_tool).
```

```
specialisation_of(create_w4_model, image_analysis_tool).
specialisation_of(create_poisson_model, image_analysis_tool).
specialisation_of(create_adaptive_poisson_model, image_analysis_tool).
specialisation_of(update_moving_average_model, image_analysis_tool).
specialisation_of(update_bg_model, image_analysis_tool).

specialisation_of(cvApproxChains, structural_analysis_tool).
specialisation_of(cvDrawContours, structural_analysis_tool).
specialisation_of(cvContourArea, structural_analysis_tool).
specialisation_of(cvFitEllipse, structural_analysis_tool).
specialisation_of(cvConvexHull, structural_analysis_tool).
specialisation_of(cvMinAreaRect, structural_analysis_tool).
specialisation_of(cvContourBoundingRect, structural_analysis_tool).

specialisation_of(cvCalcCovarMatrixEx, object_recognition_tool).
specialisation_of(cvCalcEigenObjects, object_recognition_tool).
specialisation_of(cvCreate2DHMM, object_recognition_tool).
specialisation_of(cvUniformImgSegm, object_recognition_tool).
specialisation_of(cvEstimateHMMStateParams, object_recognition_tool).
specialisation_of(cvEViterbi, object_recognition_tool).

specialisation_of(detect_moving_objects, object_recognition_tool).
specialisation_of(perform_morphological_operation, object_recognition_tool).
specialisation_of(count_fish_frame, object_recognition_tool).

specialisation_of(cvCalibrateCamera, '3D_reconstruction_tool').
specialisation_of(cvFindExtrinsicCameraParams, '3D_reconstruction_tool').
specialisation_of(cvRodrigues, '3D_reconstruction_tool').
specialisation_of(cvUndistortOnce, '3D_reconstruction_tool').
specialisation_of(cvFindFundamentalMatrix, '3D_reconstruction_tool').
specialisation_of(cvFindRuns, '3D_reconstruction_tool').
specialisation_of(cvCreatePOSITObject, '3D_reconstruction_tool').
specialisation_of(cvCreateHandMask, '3D_reconstruction_tool').

specialisation_of(cvCreateImage, basic_structures_operations_tool).
specialisation_of(cvSetImageROI, basic_structures_operations_tool).
specialisation_of(cvCopyImage, basic_structures_operations_tool).
specialisation_of(cvCreateMemStorage, basic_structures_operations_tool).
specialisation_of(cvCreateSeq, basic_structures_operations_tool).
specialisation_of(cvCreateSet, basic_structures_operations_tool).
specialisation_of(cvCreateGraph, basic_structures_operations_tool).
specialisation_of(cvAllocArray, basic_structures_operations_tool).
```

```
specialisation_of(cvAdd, basic_structures_operations_tool).
specialisation_of(cvTranspose, basic_structures_operations_tool).
specialisation_of(cvCopy, basic_structures_operations_tool).
specialisation_of(cvMahalonobis, basic_structures_operations_tool).
specialisation_of(cvLine, basic_structures_operations_tool).
specialisation_of(cvRectangle, basic_structures_operations_tool).
specialisation_of(cvAbsDiff, basic_structures_operations_tool).
specialisation_of(cvSqrt, basic_structures_operations_tool).
specialisation_of(cvFillImage, basic_structures_operations_tool).
specialisation_of(cvKMeans, basic_structures_operations_tool).

specialisation_of(view_video, basic_structures_operations_tool).
specialisation_of(preliminary_analysis, basic_structures_operations_tool).
specialisation_of(grab_frame_image, basic_structures_operations_tool).
specialisation_of(compute_write_luminosity, basic_structures_operations_tool).
specialisation_of(compute_write_clearness, basic_structures_operations_tool).
specialisation_of(compute_write_presence_fish, basic_structures_operations_tool).
specialisation_of(compute_write_presence_algae, basic_structures_operations_tool).
specialisation_of(write_frames_to_video, basic_structures_operations_tool).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% canPerform(Instance, VIP_Operation)                                        %%
%% A tool instance that can perform a VIP operation                           %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
canPerform(view_video, view_video).
canPerform(preliminary_analysis, preliminary_analysis).
canPerform(grab_frame_image, grab_frame_image).
canPerform(compute_write_luminosity, compute_write_luminosity).
canPerform(compute_write_clearness, compute_write_clearness).
canPerform(compute_write_presence_fish, compute_write_presence_fish).
canPerform(compute_write_presence_algae, compute_write_presence_algae).
canPerform(detect_moving_objects, detect_moving_objects).
canPerform(perform_morphological_operation, detect_correct_blobs).
canPerform(count_fish_frame, count_fish_frame).
canPerform(extract_RGB_colours, compute_predominant_colours).
canPerform(compute_histogram, compute_histogram).
canPerform(compute_main_statistical_moments, compute_main_statistical_moments).
canPerform(compute_gabor_filter, compute_gabor_filter).

canPerform(create_gaussian_bg_model, create_background_model).
canPerform(create_gaussian_mixture_model, create_background_model).
canPerform(create_moving_average_model, create_background_model).
```

```
canPerform(create_intra_frame_difference_model, create_background_model).
canPerform(create_w4_model, create_background_model).
canPerform(create_poisson_model, create_background_model).
canPerform(create_adaptive_poisson_model, create_background_model).

canPerform(update_moving_average_model, update_background_model).
canPerform(update_bg_model, update_background_model).
canPerform(extract_hsv_values, extract_hsv_values).
canPerform(compute_backprojection, compute_backprojection).
canPerform(compute_connected_components, compute_connected_components).
canPerform(compute_ratio_convex_hull_blob, compute_ratio_convex_hull_blob).
canPerform(compute_camshift, compute_camshift).
canPerform(compute_closest_blob, compute_closest_blob).
canPerform(detect_track, detect_track).
canPerform(compute_write_num_fish_frame, compute_write_num_fish_frame).
canPerform(compute_write_num_fish_video, compute_write_num_fish_video).
canPerform(compute_fish_blocking_screen, compute_fish_blocking_screen).
canPerform(write_frames_to_video, write_frames_to_video).
```

Bibliography

[1] Project aceMedia. *Integrating Knowledge, Semantics and Content for User-Centred Intelligent Media Services*. 2004. http://www.acemedia.org/aceMedia. Last accessed: Apr 21st, 2010.

[2] R. Allen. Workflow: An Introduction. In Layna Fisher, editor, *Workflow Handbook*, chapter 1, pages 15–38. Future Strategies Inc., 2001.

[3] L. Alvarez, F. Guichard, P. L. Lions, and J. M. Morel. Axioms and Fundamental Equations of Image Processing. *Archive for Rational Mechanics and Analysis*, 123(3):199–257, 1993.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Version 1.1 (BPEL)*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel. Last accessed: Apr 21st, 2010.

[5] G. Antoniou and F. van Harmelen. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2008.

[6] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications, 2nd edition*. Springer Publishing Company, Inc., 2008.

[7] M. Bertini, A. Del Bimbo, G. Serra, C. Torniai, R. Cucchiara, C. Grana, and R. Vezzani. Dynamic Pictorially Enriched Ontologies for Digital Video Libraries. *IEEE MultiMedia*, 16(2):42–51, 2009.

[8] A. Bevilacqua. *A System for Detecting Motion in Outdoor Environments for a Visual Surveillance Application*. PhD thesis, University of Bologna, Italy, 2002.

[9] S. Bloehdorn, K. Petridis, C. Saathoff, N. Simou, V. Tzouvaras, Y. Avrithis, S. Handschuh, I. Kompatsiaris, S. Staab, and M. G. Strintzis. Semantic Annotation of Images and Videos for Multimedia Analysis. In *2nd European Semantic Web Conference (ESWC'05)*, LNCS, pages 592–607. Springer, 2005.

[10] A. L. Blum and M. L. Furst. Fast Planning through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[11] D. T. Blumstein and J. C. Daniel. *Quantifying Behavior the JWatcher Way*. Sinauer Associates, Inc., 2007. `http://www.jwatcher.ucla.edu`. Last accessed: Apr 21st, 2010.

[12] J. Blythe, E. Deelman, and Y. Gil. Automatically Composed Workflows for Grid Environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.

[13] V. Bombardier, P. Lhoste, and C. Mazaud. Modélisation et Intégration de Connaissances Métier pour L'identification de Défauts par règles Linguistiques Floues. *Traitement du Signal*, 21(3):227–247, 2004. (In French).

[14] G. Booch, I. Jacobson, and J. Rumbaugh. *OMG Unified Modeling Language Specification*. Version 1.3 First Edition, 2000.

[15] G. R. Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface. *Intel Technology Journal*, 2(2):12–21, 1998.

[16] D. Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium (W3C), 2004. `http://www.w3.org/TR/rdf-schema`. Last accessed: Apr 21st, 2010.

[17] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Managing the Evolution of Dataflows with VisTrails. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications (Sciflow'06)*, pages 71–75, 2006.

[18] S. J. Caton, O. F. Rana, and B. G. Batchelor. Distributed Image Processing Over an Adaptive Campus Grid. *Concurrency and Computation: Practice and Experience*, 21(3):321–336, 2009. Special Issue: The Best of CCGrid'2007: A Snapshot of an 'Adolescent' Area.

[19] Y. H. Chen-Burger and F. P. Lin. A Semantic-based Workflow Choreography for Integrated Sensing and Processing. In *The 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'05)*, 2005.

[20] Y. H. Chen-Burger and J. Stader. Formal Support for Adaptive Workflow Systems in a Distributed Environment. In Layna Fisher, editor, *Workflow Handbook*, chapter 8, pages 93–118. Future Strategies Inc., 2003.

[21] S. A. Chien and H. B. Mortensen. Automating Image Processing for Scientific Data Analysis of a Large Image Database. *IEEE PAMI*, 18(8):854–859, 1996.

[22] V. Clément and M. Thonnat. A Knowledge-Based Approach to Integration of Image Procedures Processing. *CVGIP: Image Understanding*, 57(2):166–184, 1993.

[23] R. Clouard, A. Elmoataz, and F. Angot. Une Bibliothèque et un Environnement de Programmation d'opérateurs de Traitement d'images. In *Rapport interne du GREYC, Caen, France*, 1997. `http://www.greyc.ensicaen.fr/~regis/Pandore/index.html`. Last accessed: May 10th, 2010. (In French).

[24] R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg : A Knowledge-Based System for Automatic Generation of Image Processing Programs. *IEEE PAMI*, 21(2):128–144, 1999.

[25] S. Colantonio, I. B. Gurevich, M. Martinelli, O. Salvetti, and Y. Trusova. Thesaurus-Based Ontology on Image Analysis. In *SAMT*, pages 113–116, 2007.

[26] K. Currie and A. Tate. O-Plan: the Open Planning Architecture. *Artificial Intelligence*, 52(49-86), 1991.

[27] J. Daugman. Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression. *IEEE Trans on Acoustics, Speech, and Signal Processing*, 36(7):1169–1179, 1988.

[28] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. pages 99–116, 2004.

[29] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An Overview of Workflow System Features and Capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[30] E. Deelman, G. Singh, M.H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, K. Vahi, B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[31] L. di Stefano, G. Neri, and E. Viarani. Analysis of Pixel-level Algorithms for Video Surveillance Applications. In *11th International Conference on Image Analysis and Processing (ICIAP'01)*, pages 542–546, 2001.

[32] B. Draper, A. Hanson, and E. Riseman. Knowledge-Directed Vision : Control, Learning and Integration. *Proceedings of the IEEE*, 84(11):1625–1637, 1996.

[33] Ecogrid. National Center for High Performance Computing, Taiwan, 2006. `http://ecogrid.nchc.org.tw`. Last accessed: Apr 21st, 2010.

[34] R. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[35] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2003.

[36] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *14th Conference on Scientific and Statistical Database Management*, pages 37–46, 2002.

[37] A. R. J. Francois, R. Nevatia, J. Hobbs, and R. C. Bolles. VERL: An Ontology Framework for Representing and Annotating Video Events. *IEEE MultiMedia*, 12(4):76–86, 2005.

[38] Freefluo. *Freefluo Workflow Enactor Overview*. 2004. `http://www.freefluo.sourceforge.net`. Last accessed: Apr 21st, 2010.

[39] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., 2004.

[40] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for Pegasus: Creating Large-scale Scientific Applications using Semantic Representations of Computational Workflows. In *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence (IAAI'07)*, pages 1767–1774. AAAI Press, 2007.

[41] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc., 2001. http://www.petrinets.info. Last accessed: May 3rd, 2010.

[42] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Springer, 2004.

[43] GridLab. *A Grid Application Toolkit and Testbed*. 2005. http://www.gridlab.org. Last accessed: Apr 21st, 2010.

[44] Silicon Graphics Inc./Khronos Group. *Open Graphics Library (OpenGL)*. 1997. http://www.opengl.org. Last accessed: May 17th, 2010.

[45] I. Haritaoglu, D. Harwood, and L. S. Davis. W4: Real-Time Surveillance of People and Their Activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:809–830, 2000.

[46] Jörg Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3):57–62, 2001.

[47] C. Hogg, H. Muñoz Avila, and U. Kuter. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In *AAAI'08: Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 950–956. AAAI Press, 2008.

[48] L. Hollink and M. Worring. Building a Visual Ontology for Video Retrieval. In *Proceedings of the 13th annual ACM International Conference on Multimedia (MULTIMEDIA '05)*, pages 479–482. ACM, 2005.

[49] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. World Wide Web Consortium (W3C), 2004. http://www.w3.org/Submission/SWRL. Last accessed: Apr 21st, 2010.

[50] D. C. Howell. *Statistical Methods for Psychology*. Belmont, CA, 6th edition, 2007.

[51] C. Hudelot. *Towards a Cognitive Vision Platform for Semantic Image Interpretation; Application to the Recognition of Biological Organisms*. PhD thesis, Nice-Sophia Antipolis University, 2005.

[52] INRIA. *ORION: Intelligent Environments for the Resolution of Problems for Autonomous Systems*. 2007. http://www.inria.fr/recherche/equipes/orion.en.html. Last accessed: Apr 21st, 2010.

[53] Intel. *Open Source Computer Vision (OpenCV) Library*. 2006. http://sourceforge.net/projects/opencvlibrary. Last accessed: Apr 21st, 2010.

[54] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'04)*, pages 125–131. ACM Press, 2004.

[55] G. Klyne and J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium (W3C), 2004. http://www.w3.org/TR/rdf-concepts. Last accessed: Apr 21st, 2010.

[56] P. Langley and D. Choi. A Unified Cognitive Architecture for Physical Agents. In *21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1469–1474. AAAI Press, 2006.

[57] A. Lansky, M. Friedman, L. Getoor, S. Schmidler, and N. Short. The Collage/Khoros Link: Planning for Image Processing Tasks. In *Integrated Planning Applications: Papers from the 1995 AAAI Spring Symposium*, pages 67–76, 1995.

[58] A. Lee. *VirtualDub Video Capture/Processing Utility*. 2006. http://www.virtualdub.org. Last accessed: Apr 21st, 2010.

[59] N. Li, S. Kambhampati, and S. Yoon. Learning Probabilistic Hierarchical Task Networks to Capture User Preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1754–1759, 2009.

[60] C. E. Liedtke and A. Blömer. Architecture of the Knowledge Based Configuration System for Image Analysis "Conny". In *International Conference on Pattern Recognition (ICPR'92)*, pages 375–378, 1992.

[61] X. Liu, J. Liu, J. Eker, and E. A. Lee. Heterogeneous Modeling and Design of Control Systems. *Software-Enabled Control: Information Technology for Dynamical Systems*, pages 105–122, 2003.

[62] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2005.

[63] B. Ludäscher, N. Podhorszki, I. Altintas, S. Bowers, and T. M. McPhillips. From Computation Models to Models of Provenance: The RWS Approach. *Concurrency and Computation: Practice and Experience*, 20(5):507–518, 2008.

[64] N. Maillot, M. Thonnat, and A. Boucher. Towards Ontology Based Cognitive Vision. *Machine Vision and Applications (MVA)*, 16(1):33–40, 2004.

[65] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Technical report, 2004. `http://www.w3.org/Submission/OWL-S`. Last accessed: Apr 21st, 2010.

[66] Mathworks. *MATLAB - The Language of Technical Computing*. The MathWorks Inc., 1994-2010. `http://www.mathworks.com/products/matlab`. Last accessed: Apr 21st, 2010.

[67] T. Matsuyama. Expert Systems for Image Processing: Knowledge-based Composition of Image Analysis Processes. *Computer Vision, Graphics, and Image Processing*, 48(1):22–49, 1989.

[68] R. Mayer, C. Menzel, M. Painter, P. Witte, T. Blinn, and B. Perakath. *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report, Knowledge Based Systems Inc. (KBSI)*. 1995. `http://www.idef.com/IDEF3.htm`. Last accessed: Apr 21st, 2010.

[69] T. L. McCluskey, D. Liu, and R. M. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *ICAPS'03*, pages 92–101, 2003.

[70] D. McGuinness and F. van Harmelen. *OWL Web Ontology Language*. World Wide Web Consortium (W3C), 2004. `http://www.w3.org/ TR/owl-features`. Last accessed: Apr 21st, 2010.

[71] W. K. Michener, J. H. Beach, M. B. Jones, B. Ludäscher, D. D. Pennington, R. S. Pereira, A. Rajasekar, and M. Schildhauer. A Knowledge Environment

for the Biodiversity and Ecological Sciences. *Journal of Intelligent Information Systems*, 29(1):111–126, 2007.

[72] Marine Metadata Interoperability Project (MMI), 2008. http://marinemetadata.org. Last accessed: Apr 21st, 2010.

[73] MUSCLE. Multimedia Understanding through Semantics, Computation and Learning, 2008. http://muscle.ercim.eu. Last accessed: Apr 21st, 2010.

[74] G. Nadarajan. Planning for Video Processing using Ontology-Based Workflow. In *Doctoral Consortium at the 17th International Conference on Automated Planning & Scheduling (ICAPS'07)*, pages 52–55, 2007.

[75] G. Nadarajan and Y. H. Chen-Burger. Translating a Typical Business Process Modelling Language to a Web Services Ontology through Lightweight Mapping. *IET Software*, 1(1):1–17, 2007.

[76] G. Nadarajan, Y. H. Chen-Burger, and R. B. Fisher. A Knowledge-Based Planner for Processing Unconstrained Underwater Videos. In *IJCAI'09 Workshop on Learning Structural Knowledge From Observations (STRUCK'09)*, 2009.

[77] G. Nadarajan, Y. H. Chen-Burger, and J. Malone. Semantic Grid Services for Video Analysis. In *Web Intelligence Workshop on Service Composition (SERCOMP'06)*, pages 121–124, 2006.

[78] G. Nadarajan and A. Renouf. A Modular Approach for Automating Video Processing. In *12th International Conference on Computer Analysis of Images and Patterns (CAIP'07)*, 2007.

[79] D. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F.Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[80] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila. SHOP: Simple Hierarchical Ordered Planner. In *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 968–973, 1999.

[81] NeOn. *Networked Ontologies Project.* 2006-2010. http://www.neon-project.org. Last accessed: Apr 21st, 2010.

[82] NeOn. *Methodology for Building Ontology Networks.* Networked Ontologies Project, 2010. http://www.neon-project.org/nw/NeOn_Book. Last accessed: May 5th, 2010.

[83] R. Nevatia, J. Hobbs, and B. Bolles. An Ontology for Video Event Representation. In *IEEE Workshop on Event Detection and Recognition*, 2004.

[84] L. P. Noldus, R. J. Trienes, A. H. Hendriksen, H. Jansen, and R. G. Jansen. The Observer Video-Pro: New Software for the Collection, Management, and Presentation of Time-structured Data from Videotapes and Digital Media Files. *Behavior Research Methods, Instruments and Computers*, 32(1):197–206, 2000.

[85] O-Plan2. *The Open Planning Architecture, User Guide Version 3.3.* AIAI, University of Edinburgh, 1993. http://www.aiai.ed.ac.uk/project/oplan. Last accessed: May 17th, 2010.

[86] O. Oechsle and A. F. Clark. Feature Extraction and Classification by Genetic Programming. In *International Conference on Computer Vision Systems (ICVS'08)*, pages 131–140, 2008.

[87] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[88] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[89] A. N. Rajagopalan, P. Burlina, and R. Chellappa. Higher Order Statistical Learning for Vehicle Detection in Images. In *Proceedings of the International Conference on Computer Vision (ICCV'99)*, pages 1204–1209. IEEE Computer Society, 1999.

[90] A. Renouf and R. Clouard. Hermes - A Human-Machine Interface for the Formulation of Image Processing Applications, 2007. http://www.greyc.ensicaen.fr/∼regis/Hermes/index-en.html. Last accessed: May 10th 2010.

[91] A. Renouf and R. Clouard. *The Project Pantheon: Resources*. 2009. http://www.greyc.ensicaen.fr/∼regis/Pantheon/resources/index-en.html. Last accessed: Apr 21st, 2010.

[92] A. Renouf, R. Clouard, and M. Revenu. A Platform Dedicated to Knowledge Engineering for the Development of Image Processing Applications. In *9th International Conference on Enterprise Information Systems (CEIP'07)*, pages 271–276, 2007.

[93] A. Renouf, R. Clouard, and M. Revenu. How to Formulate Image Processing Applications? In *International Conference on Computer Vision Systems (ICVS'07)*, pages 10–20, 2007.

[94] J. Rios, J. Karlsson, and O. Trelles. Magallanes: A Web Services Discovery and Automatic Workflow Composition Tool. *BMC Bioinformatics*, 10(1):334, 2009.

[95] N. Russell, A. H. M. ter Hofstede, W. M. P. Van Der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPMcenter.org, 2006.

[96] E. D. Sacerdoti. A Structure for Plans and Behavior. *American Elsevier*, 1977.

[97] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, and J. Lee. *The Process Specification Language (PSL): Overview and Version 1.0 Specification, ISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD*. 2000. http://www.nist.gov/msidlibrary/doc/nistir6459.pdf. Last accessed: Apr 21st, 2010.

[98] C. Spampinato. *A Motion Detection System Based on Statistical Objects Modeling for Outdoor Applications*. PhD thesis, DIIT, University of Catania, Italy, 2008.

[99] C. Spampinato, Y. H. Chen-Burger, G. Nadarajan, and R. B. Fisher. Detecting, Tracking and Counting Fish in Low Quality Unconstrained Underwater Videos. In *3rd International Conference on Computer Vision Theory and Applications (VISAPP'08)*, pages 514–519, 2008.

[100] R. D. Stevens, A. J. Robinson, and C. A. Goble. MyGrid: Personalized Bioinformatics on the Information Grid. *Bioinformatics*, 19(1):i302–i304, 2003.

[101] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – A Distributed Job Scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, 2001. `http://www.cs.wisc.edu/condor/manual/v6.8/2_11DAGMan_Applications.html`. Last accessed: Apr 21st, 2010.

[102] TargetJr and the IUE. *VXL: C++ Libraries for Computer Vision Research and Implementation*. 2000. `http://vxl.sourceforge.net`. Last accessed: May 17th, 2010.

[103] A. Tate. Generating Project Networks. In *Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)*, pages 888–893, 1977.

[104] A. Tate. Intelligible AI Planning - Generating Plans Represented as a Set of Constraints. In *Proceedings of the Twentieth British Computer Society Special Group on Expert Systems International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES'00)*, pages 3–16. Springer, 2000.

[105] I. Taylor, E. Deelman, D. Gannon, and M. Shields. *Workflows for e-Science*. Springer, New York, 2007.

[106] I. Taylor, M. Shields, I. Wang, and A. Harrison. Visual Grid Workflow in Triana. *Journal of Grid Computing*, 3(3-4):153–169, 2005.

[107] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana Workflow Environment: Architecture and Applications. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 320–339. Springer, New York, 2007.

[108] I. Taylor, M. Shields, I. Wang, and O. Rana. Triana Applications within Grid Computing and Peer to Peer Environments. *Journal of Grid Computing*, 1(2):199–217, 2003. `http://journals.kluweronline.com/article.asp?PIPS=5269002`. Last accessed: Apr 21st, 2010.

[109] C. Town. Ontological Inference for Image and Video Analysis. *Machine Vision Appication*, 17(2):94–115, 2006.

[110] M. Uschold and M. King. Towards a Methodology for Building Ontologies. In *IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[111] W. M. P. van der Aalst and A. H. M. ter Hofstede. *Workflow Patterns Home Page*. Eindhoven University of Technology & Queensland University of Technology, 2010. http://www.tm.tue.nl/it/research/patterns. Last accessed: Apr 24th, 2010.

[112] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed Parallel Databases*, 14(1):5–51, 2003.

[113] S. van Splunter, F. M. T. Brazier, J. A. Padget, and O. F. Rana. Dynamic Service Reconfiguration and Enactment using an Open Matching Architecture. In *International Conference on Agents and Artificial Intelligence (ICAART'09)*, pages 533–539, 2009.

[114] G. von Laszewski and M. Hategan. Java CoG Kit Karajan/Gridant Workflow Guide. Technical report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[115] WfMC. *Workflow Management Coalition*. 1993-2010. http://www.wfmc.org. Last accessed: Apr 21st, 2010.

[116] S. A. White and D. Miers. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008. http://www.bpmn.org. Last accessed: May 3rd, 2010.

[117] D. E. Wilkins. Can AI Planners Solve Practical Problems? *Computational Intelligence*, 6(4):232–246, 1990.

[118] G. Yang. Process Library. *Data & Knowledge Engineering*, 50(1):35–62, 2004.

[119] H. H. Zhuo, D. H. Hu, C. Hogg, Q. Yang, and H. Muñoz Avila. Learning HTN Method Preconditions and Action Models from Partial Observations. In *International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1804–1809, 2009.

[120] Z. Zivkovic. Improved Adaptive Gaussian Mixture Model for Background Subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04)*, pages 28–31, 2004.